Learning Assembly Tasks from Human Demonstration

Aleksi Ikkala

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 12.02.2016

Thesis supervisor:

Prof. Ville Kyrki

Thesis advisor:

D.Sc. (Tech.) Joni Pajarinen

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Aleksi Ikkala

Title: Learning Assembly Tasks from Human Demonstration

Date: 12.02.2016 Language: English Number of pages: 6+88

Department of Electrical Engineering and Automation

Professorship: Automation Technology

Supervisor: Prof. Ville Kyrki

Advisor: D.Sc. (Tech.) Joni Pajarinen

This thesis presents a method for learning and reproducing assembly tasks using Learning from Demonstration paradigm and a graph representation of assembly parts and their spatial relations. We show that this graph representation combined with inexact graph matching techniques provide a framework capable of learning assembly tasks, even with uncertain information of assembly operations. In this thesis our method replicated observed assembly tasks, where Lego Quatro bricks were manipulated with pick-and-place operations.

We tested our proposed method through a series of experiments. In the experiments, a robot first observed a human teacher demonstrate an assembly task in front of a Kinect sensor. Then, the robot generated a simulation that depicted the learned assembly product. We also introduced uncertainty into the experiments by changing some of the assembly parts, or by not showing the intermediate assembly operations to the robot. In these events, the robot generated a simulated structure that was similar to the observed one. We used inexact graph matching techniques to measure the similarity between assembly structures. In the experiments our method successfully replicated a learned task when the robot was provided with a complete set of assembly parts. Also, the task was repeated relatively well when only one or two assembly parts were replaced with another type of Lego.

We conclude that our method provides a convenient platform for a more general assembly method. Also, our method is capable of "improvising" in unanticipated situations, where the robot is supplied with imperfect knowledge of the task.

Keywords: assembly, learning from demonstration, graph theory, inexact graph matching, graph similarity measure, computer vision

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Aleksi Ikkala

Työn nimi: Kokoonpanotehtävien oppiminen ihmisen havaintoesityksistä

Päivämäärä: 12.02.2016 Kieli: Englanti Sivumäärä: 6+88

Sähkötekniikan ja automaation laitos

Professuuri: Automaatiotekniikka

Työn valvoja: Prof. Ville Kyrki

Työn ohjaaja: TkT Joni Pajarinen

Tässä diplomityössä esitämme menetelmän kokoonpanotehtävien oppimiseen ihmisen havaintoesityksistä, käyttäen graafiesitystä kokoonpanossa käytetyistä kappaleista. Osoitamme, että tämä graafiesitys yhdistettynä epätarkkoihin graafinsovitusmenetelmiin mahdollistaa kokoonpanotehtävien oppimisen — myös silloin, kun kokoonpano-operaatioiden oppimiseen liittyy epävarmuutta. Tässä työssä käytimme menetelmää toistamaan kokoonpanotehtäviä, joissa Lego Quatro -palikoita liitettiin toisiinsa.

Testasimme esitettyä menetelmää useilla kokeilla. Kokeissa robotti ensin havaitsi opettajan suorittavan kokoonpanotehtävän Kinect-kameran edessä. Tämän jälkeen robotti toisti opitun tehtävän simulaationa. Lisäsimme myös epävarmuutta kokeisiin vaihtamalla kokoonpanoon tarvittavia kappaleita, tai jättämällä näyttämättä kokoonpanon välivaiheita robotille. Tällöin robotti tuotti simuloidun rakennelman, joka oli mahdollisimman samankaltainen opittuun rakennelmaan nähden. Rakennelmien välistä samankaltaisuutta mitattiin epätarkoista graafinsovitusmenetelmistä tutuilla samankaltaisuusmitoilla. Kokeissa menetelmämme toisti opitut tehtävät onnistuneesti silloin, kun robotille oli annettu sama joukko palikoita kuin mitä havaintoesityksessä oli käytetty. Robotti toisti tehtävät menestyksekkäästi myös silloin, kun vain yksi tai kaksi havaintoesityksessä käytettyä Legoa oli vaihdettu erityyppisiksi.

Kokeiden perusteella toteamme, että esittämämme menetelmä luo hyvän pohjan yleisemmälle kokoonpanomenetelmälle. Menetelmämme pystyy myös ”improvisoimaan” odottamattomissa tilanteissa, joissa robotille näytettyyn havaintoesitykseen liittyy epätäydellistä tietoa.

Avainsanat: kokoonpano, näyttämällä oppiminen, graafiteoria, epätarkka graafinsovitus, graafien samankaltaisuusmitta, konenäkö

Preface

This thesis presents the work I did in the Intelligent Robotics group under the supervision and guidance of Ville Kyrki and Joni Pajarinen. I want to thank you both for the incessant help and support you provided when I encountered issues and obstacles. You listened to my ideas, challenged them, and forced me to reconsider my views, thus refining them into more robust ones.

Also, I want to thank those who have supported me outside of work — my family and friends. I am grateful for Tuuli and Marko, who endured my programming related whines during our (nearly) daily lunch sessions. I'm especially thankful to Tuuli; you always cheer me up when I need it.

This thesis is dedicated to my parents. For as long as I can remember, you have stressed the importance of education and pushed me forward.

Espoo, 12.02.2016

Aleksi I. Ikkala

Contents

Abstract ii

Abstract (in Finnish) iii

Preface iv

Contents v

Abbreviations & Symbols vi

1 Introduction 1

2 Background 3
2.1 Assembly . . . 5
2.1.1 Learning Spatial Relations . . . 5
2.1.2 Planning Assembly Sequences . . . 7
2.2 Learning from Demonstration . . . 9
2.2.1 Trajectory Learning . . . 11
2.2.2 Symbolic Learning . . . 11
2.2.3 Learning from Demonstration in Assembly Tasks . . . 13
2.3 Inexact Graph Matching . . . 15
2.3.1 Spectral Methods . . . 17
2.3.2 Inexact Graph Matching in Assembly Tasks . . . 21

3 Teaching and Reproducing Assembly Tasks 24
3.1 Abstraction of Object Compositions into Graphs . . . 25
3.2 Learning Symbolic Assembly Tasks . . . 27
3.2.1 Recognition and Tracking of a Lego Quatro . . . 28
3.2.2 Database of Assembly Operations . . . 29
3.2.3 Learning Pipeline . . . 31
3.3 Reproducing Symbolic Assembly Tasks . . . 35

4 Experiments and Results 41
4.1 Exact Replications . . . 42
4.2 Inexact Replications . . . 45
4.3 Greedy Construction . . . 60

5 Discussion 64
5.1 Learning Phase . . . 64
5.2 Replication Phase . . . 66
5.2.1 Inexact Replication Experiments . . . 66
5.2.2 Greedy Construction Experiments . . . 72

6 Conclusions 75

References 79

Abbreviations & Symbols

Abbreviations

AG	Assembly graph
APO	Assembly-plan-from-observation
CVFH	Clustered viewpoint feature histogram
DMP	Dynamic movement primitive
DOF	Degree of freedom
GMM	Gaussian mixture model
HMM	Hidden Markov model
IL	Imitation learning
JoEig	Joint Eigendecomposition
LfD	Learning from Demonstration
LWL	Locally weighted learning
MLS	Moving least squares
OUR-CVFH	Oriented, unique, and repeatable clustered viewpoint feature histogram
PbD	Programming by Demonstration
PFH	Point feature histogram
RGB-D	A combined color and depth image or video
RL	Reinforcement learning
SGURF	Semiglobal unique reference frame
SHOT	Signature of histograms of orientations
VFH	Viewpoint feature histogram
VSL	Visuospatial skill learning

Symbols

$\boldsymbol{A}$	An adjacency matrix of a graph
$\delta_{JoEig}(G, H)$	Dissimilarity between graphs G and H with JoEig dissimilarity measure
$\delta_{Laplacian}(G, H)$	Dissimilarity between graphs G and H with Laplacian dissimilarity measure
$diag(\boldsymbol{M})$	A vector containing the diagonal elements of matrix $\boldsymbol{M}$
$G = (\mathcal{V}, \mathcal{E})$	A graph G with vertices $\mathcal{V}$ and edges $\mathcal{E}$
$\boldsymbol{L}$	A Laplacian matrix of a graph
$\boldsymbol{\mathcal{L}}$	A symmetric normalized Laplacian matrix of a graph
$\boldsymbol{\mathcal{T}}_b^a$	A transformation matrix from b to a

1 Introduction

Industrial robots have been used extensively in manufacturing. From the largest vehicles to smallest electronic devices, more and more products are constructed with the aid of robots. These robots execute relatively simple assembly operations in specific and familiar factory settings. Typically, there is no room for errors, as the trajectories of the operations are preprogrammed in a point-by-point basis: Everything must be in precisely correct positions to avoid collisions and clenching of assembly parts. Furthermore, robots are becoming more common, and they can already be found outside factories in everyday locations, such as households, hospitals, and care homes. In order to make robots more broadly available and accessible by non-professionals, controlling and operating them must be made easier. One prospective means to achieve this objective is to enable a robot to learn how a certain movement or skill is attained, instead of manually programming the desired skill into the robot's software. Accordingly, the teacher does not need significant programming capabilities, if at all, to instruct a robot how to perform novel actions. This approach is also applicable to the assembly tasks: The learned skill could be, for instance, a series of assembly operations. If the robot learns the *skill* itself, it can be conveniently utilized in new and unpredictable settings. Thus, such an assembly skill would be more robust, as the robot could function with imperfect knowledge of the work environment and assembly parts.

The above-mentioned approach is an area of study known as the *Learning from Demonstration* (LfD)[1] paradigm. The aim of the LfD paradigm is to produce robots capable of learning new skills by observing a teacher's demonstration. The teacher demonstrates a movement — or a combination of movements — to a robot, which perceives it using an array of sensors. The teacher is most often a human, but it can be, for instance, another robot or a simulated planner. An algorithm then recognizes and extracts the salient features from the demonstration, and attempts to learn them. After learning, the robot should be able to generalize the acquired skill to new situations where environmental conditions may have changed. Moreover, the robot should also handle unexpected situations where minor errors or perturbations might occur. Billard et al. [1] and Argall et al. [2] provide comprehensive surveys of the LfD approach.

The demonstration can be executed and recorded in various ways. The precise manner in which a movement is recorded varies greatly across approaches: For example, the robot might be using its own sensors to record its own actions while being passively teleoperated by the teacher. On the other extreme, the robot might utilize a camera recording of a human demonstrator to learn the desired behaviour. In this thesis we propose an assembly method based on the latter approach. In our method, a robot first observes a human teacher demonstrate an assembly task in front of a Kinect sensor, and then replicates the observed task using a set of assembly parts assigned to it. The assembly parts may be located anywhere in the Kinect's field

[1] Also known as *Imitation Learning* (IL) and *Programming by Demonstration* (PbD).

of view during the demonstration and replication: The absolute locations of the parts are irrelevant, as we learn only the configurations of the assembly parts, that is, their positions relative to each other in the observed assembly product. All that is required to learn an assembly task is a Kinect and 3D models of the parts. Our ultimate objective is to provide a general assembly method, that could be utilized in all types of assembly tasks. This thesis shows that the proposed concept works with Lego bricks as assembly parts, and pick-and-place type of assembly operations.

We use graphs to model the structure and spatial relations of the assembly parts. The graph representation provides a compact and intuitive means of describing the parts and operations. Furthermore, the graph representation allows us to model and learn the tasks on a higher level, as opposed to learning just the trajectories of the operations. Ideally, we would like to have a graph representation that could be applied to multiple assembly operations. For example, even though we teach a robot to replicate assembly operations using Lego bricks, in an ideal situation the same scheme could be used to teach a robot to replicate assembly operations on, say, car engines. Additionally, we want the robot to replicate an assembly task, even if it is provided with uncertain or imperfect knowledge of the assembly operations used in the task. We attack the problem of uncertainty by employing *inexact graph matching* techniques. Using these techniques, the robot attempts to build a similar assembly product even if it is supplied with a set of unobserved assembly parts, or if it is not shown the intermediate assembly operations that produce the assembly product. We use two different dissimilarity measures to evaluate the similarity between assembly structures.

This thesis introduces our proposed method for learning and replicating observed assembly tasks using the LfD paradigm and a graph representation of the assembly parts. Both the LfD paradigm and the graph representation have been used in assembly task learning to some extent (for example, [3, 4]). Graphs have been typically used to model assembly operations (for instance, [5]), and less to model the structure of assembly parts. By modelling the structures with graphs, inexact graph matching can be utilized to search for similar assembly structures (see, for example, [6, 7]). We have designed a series of experiments to test and highlight the properties of our approach for learning and replicating tasks. We study also how well the chosen inexact graph matching techniques perform in assembly tasks with imprecise knowledge — that is, missing assembly parts, or undisclosed assembly operations. Additionally, we hope to provide insight to what kind of characteristics a desirable similarity measure might have.

The rest of this thesis is partitioned into five sections. First, Section 2 provides the background and context for our work. Then, Section 3 discusses in detail how our proposed assembly method functions. In Section 4 we present the experiments and results, and in Section 5 we discuss the results and related problems in detail. Finally, Section 6 concludes this thesis.

2 Background

The last five decades have seen vast developments in the manufacture of consumer products in industrial settings. These developments are largely due to the increase of robotic workforce in industries. In 1962, the first industrial robot Unimate appeared in a General Motors automobile factory (see Figure 2.1). Its job description included relatively simple tasks, such as spot welding and extracting die castings [8]. As the first of its kind, Unimate was rather clumsy, and barely a hint of what was to come. All the same, it was able to complete its responsibilities, and release human workers from their tedious and unpleasant jobs. Soon more specialized and enhanced industrial robots surfaced, and they began to find their places amongst the human workers in factory settings. More and more tasks were assigned to the robots, as they were able to perform even the most laborious, dull, and dangerous jobs.

With the industrial robots growing in number, the production rates of factories increased substantially [9]. Today most factories, irrespective of their sizes, employ robots for manufacturing products, and it is not rare to find fully automatized assembly lines. Apart from maintenance and reprogramming breaks, the robots run practically non-stop. In traditional manufacturing, experts program the robots to execute given tasks in a step-by-step manner. This means that all the knowledge of the required tools and the environment must be hardcoded into the program prior

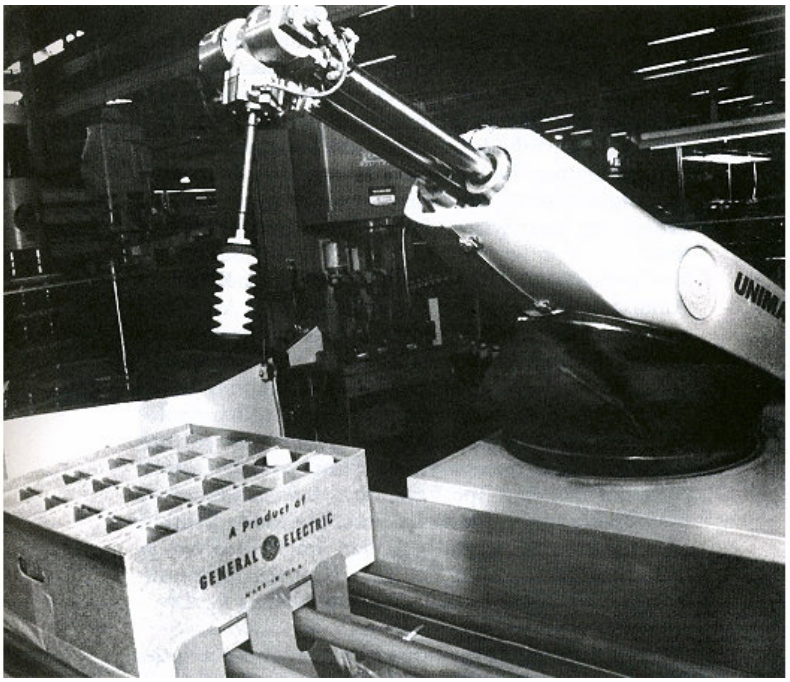


Figure 2.1: *The world's first industrial robot Unimate. The Unimate consisted of a large computer-like box and was connected to an arm. It had four programmable axes with systematic tasks stored in a drum memory. Image from* `http://www.computerhistory.org/revolution/artificial-intelligence-robotics/13/292/1272`.

to the execution of the task. This, however, unfolds a serious downside associated with the traditional manufacturing: Changes in the task settings usually require reprogramming. Reprogramming, on the other hand, is time consuming, and requires specialized labor in the form of trained programmers.

Moreover, robots are not met exclusively in industrial settings any longer. The advances in service robotics have brought robots closer to common people, that is, non-professional users. With the proliferation of end-users without programming training, the need for more autonomous robots has increased.

One approach to further the independence of robots and reduce the necessity of programming is the Learning from Demonstration (LfD) framework (see [1] or [2] for an extensive introduction to the subject). In LfD, the aim is to allow a robot to learn a desired behaviour from observations of a human's performance, and hence no programming is required. In the scope of this thesis, we categorize the LfD methods into trajectory and symbolic learning approaches. The former approaches are focused on learning the typical trajectory of, for example, a tool in a given task. The latter approaches, on the other hand, address the problem of learning a skill on a higher symbolic level, where the actions are modelled using predefined symbols. The symbolic learning approaches provide a plausible means to learn a complex skill, such as an assembly skill [10].

Assembly is a skill that can be applied in both industrial and social settings. Traditionally, an assembly task is understood in the industrial sense, where a product is manufactured following a series of assembly operations. Examples of such tasks include vehicle and small electronics assembly [11]. However, assembly is a more general concept. Many of the mundane chores one encounters every day — for instance, preparing a meal or putting dishes in a dishwasher — can be considered as assembly tasks. Just as in assembly tasks, in these chores it is important that the necessary operations are performed in the correct order, while minding the spatial relations of the objects.

In this thesis, we propose a framework for a general assembly skill. We use the LfD paradigm to teach a robot how to perform assembly tasks. First, in the learning phase, a human demonstrates the task in front of a Kinect RGB-D sensor. This task is then segmented into individual assembly operations, which depict how the assembly parts should move with relation to one another at each stage of the task. In this thesis we use Lego Quatro bricks as the assembly parts, but the method can be generalized to other types of objects as well. We depict the assembly parts using a graph representation: The parts themselves and their spatial relations are represented as graphs. Then, in the replication phase, we provide the robot with a set of Legos. However, this set of Legos may or may not be the same as was used in the demonstration. The robot attempts to either replicate the task *exactly*, if it has an identical set of bricks available, or *inexactly*, if it is provided with a different set of Legos. In the inexact replication the robot attempts to build a similar kind of structure, exploiting the learned knowledge of how the structure *should* look like after each stage in the task. We employ inexact graph matching techniques to measure

the similarity between different structures. We will generally refer to these inexact graph matching techniques as *dissimilarity measures*.

In this section we first discuss the properties of assembly task in Section 2.1: How to consider the spatial relations of assembly parts and how to plan the correct sequence of assembly operations. In Section 2.2 we go through different forms of LfD techniques and examine how LfD has already been used in assembly tasks. Finally, in Section 2.3 we discuss the theory behind inexact graph matching, present the dissimilarity measures, and review the use of inexact graph matching in assembly tasks.

2.1 Assembly

In essence, assembly is manipulation of objects augmented with an apprehension of how the objects affect each other. Assembly tasks also require some sort of planning, as usually assembly parts must be connected in a predefined manner to enable the functional purpose of the object. In addition, complex object manipulation typically calls for both motion generation on a geometric level, as well as sequential composition and reasoning on a more abstract level [12].

In [13] Kroemer and Peters discuss the interaction between a robot and objects in a manipulation task as follows: There "nearly always" exists a direct physical contact between a robot — or a manipulator in general — and an object. The contacts are divided into two categories: Contacts between the assembler and an object, and contacts between the objects in the manipulator's environment. Different types of interactions can be imposed on the objects depending on the site of the contact. For instance, a contact on the side allows a manipulator to push and slide an object, while a contact on top or bottom can be used to support an object. By combining several interactions in a suitable sequence, more intricate manipulation operations can be conducted. As Kroemer and Peters emphasize, a manipulator must be able to determine the possible interactions between objects and utilize them in order to accomplish a manipulation task. All of the above holds true for assembly tasks as well.

2.1.1 Learning Spatial Relations

In order to execute complex assembly tasks, a robot must be capable of learning symbolic representations of geometric relations between objects. For example, a robot must be able to learn that object A is required to be on top of object B, or that object C must be placed inside object D.

Kroemer and Peters [13] discuss a learning approach which utilizes contact information between assembly parts. In their method, a robot learns to detect interactions between objects by considering the objects' contact distribution. This contact distribution is computed from the contact points of the objects, that is, from the regions where the objects are in contact with each other. The resulting point cloud of contact

points is then modelled as a Gaussian distribution, such that different contact types have different distributions. After the robot has been taught with different kinds of contacts, Kroemer and Peters employ a Bhattacharyya kernel function [14] to compute the similarity between a previously unseen contact distribution and the set of known contact distributions. The robot uses similarity measures and kernel logistic regression to classify the potential interaction into one of the known interaction types. This method is successfully utilized to teach a robot how to build block towers using blocks of different shapes. In this thesis we also consider the contacts between Lego bricks to learn plausible configurations. However, we define the contacts as links in a graph, and, instead of kernel functions, we compute the similarities between observed configurations and unseen configurations by inexact graph matching techniques. Furthermore, we do not classify these unseen configurations into contact types, we simply measure the similarity between them and the observed configurations. In our work, the similarity is maximized to find an assembly operation that resembles most the one demonstrated by the human teacher. The manner in which the similarities are computed is explained in Section 2.3.

Rosman and Ramamoorthy [15] also use contact points to learn spatial relations between objects, but instead of contact distributions, they use contact networks. The contact point networks are topological representations of objects in a stereo image. Effectively, the network representations are used to abstract the objects into graph-like skeletons, which are then used to identify areas where an object touches another object. This type of structure is a useful abstraction for manipulation, as it reveals the points where an object interacts with the external world or another object. Furthermore, this representation also aids in classifying the relationships between the objects, and Rosman and Ramamoorthy use a *Support Vector Machine* (SVM) to classify the relationships between objects. However, these relations are confined to simple "on top of," "next to," and "both on top of and next to" another object. Again, we are not interested in specifically categorizing the types of relations between two objects. Instead, we are interested in the overall similarity between two configurations of assembly parts. Moreover, in the approach of Rosman and Ramamoorthy, the graphs are formed according to where the objects are in close contact with each other, whereas in our proposed method the objects are abstracted into graphs before any assembly operations. In other words, the nodes in our graphs depict areas where two objects can be joined, instead of areas where two objects are already joined or are in close contact. The abstraction of the assembly parts into graphs is discussed in Section 3.1.

Kulick et al. [12] investigate an interactive teaching scenario, where a human teacher demonstrates the geometric properties of objects to a robot. The objective is to efficiently learn symbolic representations of the relations between objects by actively generating pick-and-place geometric situations with different objects. Kulick et al. also demonstrate that the learned symbols can be used in a relational *Reinforcement Learning* (RL) framework to learn probabilistic relational rules and use them to solve object manipulation — or basically assembly — tasks in a goal-directed manner. The purpose of active learning is to reduce the amount of training data and subsequently

reduce the human's burden in teaching symbols. The teaching method proposed in this thesis is flexible enough to allow implementation of interactive teaching. Currently, however, this form of interactive teaching is omitted from our method.

A distinctive approach to learning spatial relations was proposed by Sjöö and Jensfelt [16]. According to Sjöö and Jensfelt, human-inhabited spaces are structured by patterns of functional relationships: Buildings, utensils and furniture alike have specific functional purposes. Humans are well aware of these functional purposes, and can operate in such environments. As service robots work in the same environments as humans, they should also be aware of the spatial functional relationships. Sjöö and Jensfelt discuss how the apprehension of these functional relationships enables a robot to

- *Decompose a process, such as an assembly task, to relevant aspects.* For instance, if a robot observes a complex assembly task, it is able to divide it into abstract, generalizable steps, such as "A goes *on top* of B, then B goes *through* C."
- *Apply abstract task knowledge to novel objects and situations.* A robot knows, for example, that if "I put A on top of B, and then I *move* B, A also moves."
- *Store and process spatial knowledge efficiently.* For instance, instead of storing trajectories of subtasks or exact metric distances of objects, a robot is able to represent only qualitative transitions such as an object being put in a certain place.

A robot should possess these qualities to effectively complete assembly tasks. The principles of symbolic learning and accomplishing assembly tasks are further discussed in Sections 2.2 and 2.2.3.

In addition to being interested in how diverse objects relate to each other spatially, we are also interested in the way in which these objects and their structure can be represented. An intuitive choice could be a graph representation, as was already discussed regarding the work of Rosman and Ramamoorthy [15]. This representation can be abstracted into topological features, which can be efficiently exploited in manipulation and planning. The use of graph structures in assembly tasks is further discussed in the coming sections.

2.1.2 Planning Assembly Sequences

Another major problem in assembly concerns finding a feasible sequence of assembly operations to reach the final assembly product. The earliest reasoning systems had to rely on external help from humans in order to generate a valid sequence [17]. These systems were based on user interaction, and they queried the user for information in geometric reasoning or precedence relations. Then, with the help of the answers, the assembly systems generated the correct sequences [18, 19]. Unsurprisingly, the automatic generation of assembly sequences is an extremely difficult problem for a planning system — Kavraki and Kolountzakis [20] proved that even in a simple

two-dimensional assembly the worst-case computational complexity of generating assembly sequences is NP-complete.

The compact structure of a graph is well suited for planning assembly sequences. de Mello and Sanderson [5] introduced the *AND/OR Graph* (AOG) representation to find the most convenient plan to efficiently assemble a product [21, 22]. The work of de Mello and Sanderson on the subject of disassembly/assembly has been further developed by a number of authors, for instance, [23] developed a CAD-directed automatic assembly sequence planning using a graph-based approach, and [24] presented an integrated computer aid for generating assembly sequences for mechanical products. Likewise, the AOG representation has been used frequently in assembly/disassembly scheme, for example [25] used it for end-of-life disassembly of electronic equipment, and similarly [26] used AOGs to determine how used products are disassembled in a cost-effective manner to obtain useable components.

Mosemann and Wahl [17] and Thomas and Wahl [27] also use AOGs to represent assembly plans, but they approach the problem of planning assembly sequences from a different angle than de Mello and Sanderson [5]. Specifically, they present a method to decompose a complex sequence of assembly operations into trajectory-based skill primitives. They classify the type of assembly operation by analysing the symbolic spatial relations between the assembly parts, the depart-spaces and necessary tools. The downsides of the method proposed by [17] are that it requires user defined CAD data and symbolic spatial relations of the assembly parts, as well as a description of the robot's workcell. The assembly method proposed in this thesis requires only models of the assembly parts, and that the user defines how the models are abstracted into a graph structure. Also, contrary to [17, 27], the application of any kind of trajectory-based skill primitives (for instance, *Dynamic Movement Primitives*, DMP), although much used in task learning[2], was deemed unnecessary in the scope of this thesis.

Ferch et al. [6] presented a method for planning assembly sequences for multiple robots. Conventionally, when using multiple robots for assembly, one uses planning software to determine a practical sequence of actions for each robot. Ferch et al., however, use the robots as independent cognitive units, and find an assembly sequence for every robot with a minimum expense of planning. The algorithm Ferch et al. use is trial-and-error based Q-learning [32], and they use a dynamic graph structure to represent the states and actions. Moreover, just as the assembly method proposed in this thesis, their algorithm does not require strict copies of subassembly parts while replicating a learned task. Their method finds a similar assembly product by using graph matching with edit distance as the measure of similarity, whereas the (dis)similarity measures we use are based on the graphs' spectra.

Unlike in the aforementioned publications, we use graphs to represent the structure of an assembly product instead of searching for assembly plans. More specifically, we do not have a planner to design an assembly sequence: Instead, we use greedy

[2]See Section 2.2.1 and, for example, [28–31].

planning[3]. As the robot replicates an observed demonstration, it simply chooses the best action at each stage of the assembly: The best action maximizes the similarity between the created structure and the learned structure.

2.2 Learning from Demonstration

Argall et al. [2] and Billard et al. [1] provide excellent surveys of the Learning from Demonstration framework. They explain the motivation behind LfD, and report where and how the framework can be applied. In this section we follow primarily the account of Argall et al. on the subject.

The world is convoluted and intricate. In order to function in such surroundings, a robot must be able to learn a mapping between the world states and its own actions. This mapping, also called a policy, dictates which actions a robot must undertake when it observes a particular world state. Instead of handcrafting policies that are general and sensible enough for a robot to use, we typically apply machine learning techniques to obtain such mappings.

The objective of LfD is to transfer skills from a human teacher to a robot via demonstrations performed by the teacher. These demonstrations are comprised of state-action pairs which tell us how a robot should act in certain situations. The state-action pairs are recorded into a dataset of examples, which is subsequently utilized by LfD algorithms to provide a policy which leads to desired robot behaviour.

There are different means to collect these demonstrations. In general, the methods can be divided into two categories depending on what the teacher uses as a platform for the execution. If the demonstration is performed on the actual robot learner (or a physically identical platform), the demonstration falls into the *demonstration* category. Then, on the other hand, if the demonstration is performed on a platform which is not the robot learner (or a physically identical platform), it belongs to the *imitation* category. These categories can be divided into more specific subcategories depending on how the demonstration is recorded (see Figure 2.2). These subcategories include *teleoperation*, *shadowing*, *sensors on teacher* and *external observation* [2].

Teleoperation In teleoperation the robot records its own sensors while being teleoperated by the teacher. Teleoperation provides the most direct method for information transfer within demonstration learning. A variant of teleoperation is *kinesthetic teaching*, where the robot is passively moved by the teacher while it records its own sensors.

Shadowing In shadowing the robot mimics the teacher's demonstration while recording its own sensors. In comparison to teleoperation, shadowing requires

[3]One might argue, however, that some sort of implicit planning is learned from the demonstrations. As the human builds the Legos step-by-step according to a plan, the robot learns which Legos he should use at each stage of the assembly.

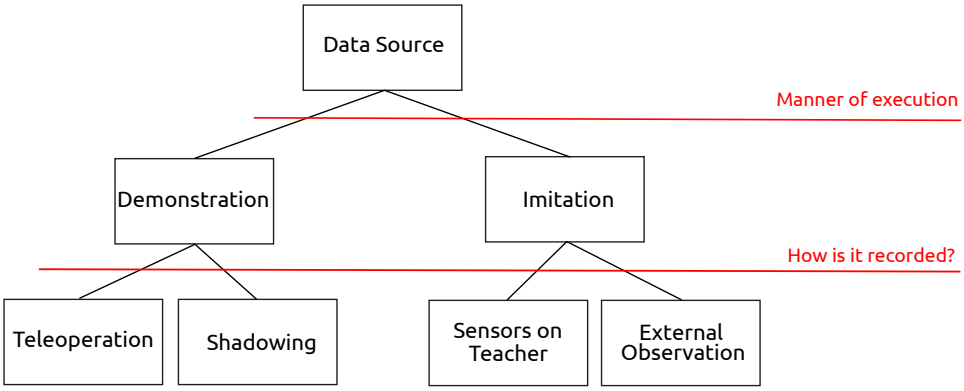


Figure 2.2: *The division of LfD methods, adapted from [2]. The methods are first divided according to what the teacher uses as a platform for execution, and further divided according to how the measurements are recorded.*

an extra algorithmic component which enables the robot to track and actively shadow the teacher.

Sensors on Teacher In this approach the recording sensors are directly on the executing platform, the teacher. These sensors may be, for example, human-wearable motion suits. However, these possibly specialized sensors limit the applicability of this technique.

External Observation In this method we rely on data recorded from sensors located external to the executing platform. Typically the external sensors are vision based, such as RGB-D cameras.

In this thesis we employ the last approach, where the robot learner observes a human performing assembly operations through a Kinect sensor. External observation is the most general approach, but it provides less reliable measurements since the robot has to rely on its imperfect sensors and indecisive software.

Nonetheless, in the scope of this thesis, we divide the methods more broadly into trajectory and symbolic, or goal-based, learning as according to [33, chap. 1.4.2]. The objective of trajectory learning is to learn tangible physical quantities, which are mapped into the physical world. A good example of such physical quantities are the (x,y,z) coordinates of an end effector — or alternatively, the joint positions — while a robot arm executes a task such as wood planing [34]. As in [34], kinesthetic teaching is often used in trajectory learning. However, the robot must be manageable in this type of teaching . The robot has to be relatively small and lightweight, and equipped with a gravity-compensation controller so that the teacher can effortlessly move the robot around. Nevertheless, trajectory learning can be used to teach fairly complex skills, at least when provided with the capability to learn the exerted forces during the demonstrations. Examples of such tasks are the wood planing mentioned above [34], and ironing and door opening tasks [35].

In this thesis, however, we want the robot to learn more abstract concepts than plain trajectories. In many tasks learning the trajectories is not enough. Instead, symbolic level knowledge is required to learn a task. An example of this type of task is solving a jigsaw puzzle: Learning of every trajectory in such complex tasks increases the complexity of the learning process unnecessarily [36]. More specifically, in this thesis the assembly parts — the blocks of Legos — are represented as graphs, and we want the robot to learn how to connect separate graphs together appropriately. The appropriateness of the assembly operations is measured by comparing the replicated Lego structure to the one built by the human teacher. As mentioned earlier, our approach generalizes to other assembly parts and operations as well, as long as the parts are representable in a graph form. These requirements, and the abstraction of assembly parts into graphs, are discussed in detail in Section 3.1.

2.2.1 Trajectory Learning

In trajectory learning the aim is to utilize multiple demonstrations in order to learn a generalized trajectory for a task. These trajectories can be recorded in joint space, task space or torque space [1], and some tasks — such as in-contact tasks — may require integration of additional information, such as information about exerted forces (for example, [34, 35]).

The trajectories are generally represented as probability distributions or dynamic models [37]. In the former approach the idea is that we compute the mean and variance of multiple trajectories, and the variance then tells us to what extent the learned trajectory can fluctuate around the mean trajectory. Hence, as the variation between the demonstrations is encoded in the variance, the individual demonstrations can be quite different. There are various means of modelling the probability density functions of trajectories, such as *Gaussian mixture models* (GMM; for example, [38, 39]), *locally weighted learning* (LWL; [40, 41]), and *hidden Markov models* (HMM; [42, 43]).

On the other hand, the learned trajectories can be modelled as dynamical models. These are designed to be stable and robust against perturbations. A widely used model in LfD are the *dynamic movement primitives* (DMP; for example, [44, 45]). DMPs are movement sequences, or "units of action," that are highly flexible in creating complex behaviours that can be adapted to dynamically changing stochastic environments [46]. DMPs model attractor behaviours of autonomous nonlinear dynamical systems with the help of statistical learning techniques [47].

2.2.2 Symbolic Learning

The objective of symbolic learning is to learn actions identified with a set of predefined symbols. These symbols are then used to encode a given demonstration. As the ultimate aim of LfD is to learn complex high-level tasks — such as loading dishes

into a dishwasher [48] — it seems necessary for a robot to be able to learn abstract concepts, which the framework of symbolic learning enables. The framework allows a robot to learn the hierarchy between actions, and also provides a set of rules which the actions must obey.

Ekvall and Kragic [49] describe the principles of symbolic learning in a form of a general task learner (see also Figure 2.3). A learner is first shown multiple demonstrations of a task, which is then decomposed into specific actions or *primitives*. These primitives are modelled as states — or symbols — which describe the impact of a certain action to the current world state. As the task as a whole is unlikely to be observed again, these states enable a robot to generalize across multiple demonstrations. In the generalization phase the robot also must consider which states have to occur, and in what order, to be able to reach the goal of the task. The constraints, which determine the order of the states, are extracted from the demonstrations, and the number of constraints also depend on the number of observed demonstrations. Ultimately the robot plans and executes a sequence of actions which accomplish the task. However, in our method a task is demonstrated only once, and the robot must follow the same order of the states, the intermediate steps of the assembly, as the teacher exhibits.

Ahmadzadeh et al. [50] use a similar type of approach for symbolic goal-based learning. Their suggested method uses imitation learning to learn primitive actions, in other words, trajectory-based skills. They employ a *visuospatial skill learning* (VSL) module to capture the spatial relationships between an object and its surrounding objects. The VSL uses visual perception to extract a sequence of actions which is used to reach the goal of a given task. A task is then represented in a generalized, discrete symbolic form.

The intent of high-level learning is to learn tasks as a sequence of abstract high-level symbols. As a downside, this type of learning often requires an abundance of prior knowledge of the objects' properties [33, chap. 1.4.2][36]. One attempt to overcome this problem was presented by Abdo et al. in [51], where they augmented higher-level learning with low-level motor commands. Also, according to Abdo et al., computing solutions solely based on the low-level motor commands is infeasible due to the high-dimensionality of the resulting problem of planning correct movements. Abdo et al. propose a learning process that allows the robot to learn new actions, such that it can later reason about the actions on both the motion level and the symbolic level. However, their suggested method assumes that the preconditions and effects of an action can be expressed using a set of predefined features. Hence, the robot can observe the world only through these handcrafted features, and thus the problem of prior knowledge is not completely solved.

As was mentioned earlier, our approach requires prior structural knowledge of the assembly parts. First of all, we need models of the objects in order to recognize and track them during the demonstrations of assembly tasks. Secondly, the models must be abstracted into graphs by a human. All assembly operations are defined in terms of these graphs: We use the graphs to find the correct assembly operation at

each stage in the task, and we use the graphs to compare the replicated structure to the learned one. Lastly, in this thesis, we use only simple pick-and-place type of assembly operations. If we wanted the method to learn different types of operations, we should define how these operations are represented and encoded into the system.

The assembly operations are encoded using the graph representations of the assembly parts, or more specifically, the *adjacency matrices* of these graphs. Each assembly operation is defined in terms of three adjacency matrices and a transformation matrix: The adjacency matrices of the initial assembly parts, the transformation matrix between them in the structure after the operation, and the adjacency matrix of this structure. These matrices form the set of symbols which we use to encode each assembly task. In Section 3.1 we discuss in detail how the assembly parts are abstracted into graphs. Section 3.2.2 describes how the assembly operations are encoded into the database of learned operations.

2.2.3 Learning from Demonstration in Assembly Tasks

Assembly tasks often require higher-level symbolic learning, and provide reasonably challenging problems in the LfD framework. In assembly tasks, learning only trajectories is usually not sufficient, and some kind of planning or symbolic learning is required. Take, for instance, the jigsaw puzzle example mentioned earlier. This puzzle can be regarded as an assembly task, and, as we already noted, learning all the required trajectories to complete the puzzle increases the complexity of the learning process unnecessarily. However, with symbolic learning, the puzzle can be completed by learning the spatial relations of the puzzle pieces.

Already in early 1990's a similar scheme as LfD was discussed by Ikeuchi and Suchiro in their endeavour to learn assembly operations. In [52] Ikeuchi and Suchiro present the *assembly-plan-from-observation* (APO) method, which aims to build a system that has the ability to observe a human execute an assembly task, understand the properties of the task, and generate a program to achieve the same task. In the APO system a human demonstrator performs the task in front of a video camera. This continuous sequence of images is then subjected to six operations:

- *Temporal Segmentation* — The video is partitioned into meaningful segments which correspond to separate human assembly tasks.
- *Object Recognition* — Objects and their poses are recognized in a given image segment.
- *Task Recognition* — The assembly task is recognized based on the results of object recognition.
- *Grasp Recognition* — Recognize where and how the human demonstrator grasps an object.
- *Global Path Recognition* — Recognize the path along which the demonstrator moves an object while avoiding collisions.

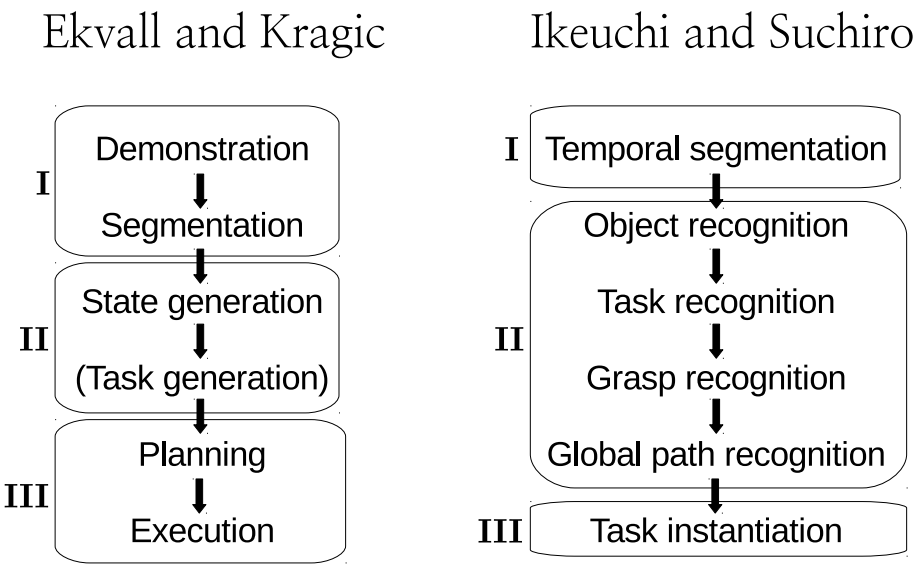


Figure 2.3: *Principles of symbolic learning by Ekvall and Kragic on the left, and stages of assembly learning by Ikeuchi and Suchiro on the right. Roman numerals depict the parts of these approaches that are essentially equivalent. Adapted from [49] and [52].*

- *Task Instantiation* — Use the information collected in previous steps to plan and perform the same assembly task.

The above list describes the necessary stages that must be addressed in order to learn an assembly task from demonstration. Note how these stages by Ikeuchi and Suchiro relate to the principles of symbolic learning by Ekvall and Kragic discussed in Section 2.2.2. As Figure 2.3 demonstrates, the approaches are quite similar. This is not surprising, since learning an assembly task is essentially symbolic learning. In Figure 2.3 the pipelines for both processes have been divided into equivalent partitions, indicated by the Roman numerals. Partition **I** deals with the segmentation of a demonstration into coherent subtasks, where as partition **II** determines "which object is moved where, and how." We have ignored the *Task generation* step from the principles of Ekvall and Kragic, because it has to do with the generalization of a task, which is not discussed by Ikeuchi and Suchiro. Lastly, in partition **III** the observed task is planned and executed. In Section 3 we go through these partitions and discuss how they are implemented in this thesis.

We want a robot to learn an abstracted and generalizable skill that could be applied to never before seen situations, possibly even with never before seen assembly parts. As already discussed in Section 2.1, the usage of graphs is a viable candidate to learn such generalizable skill representations. The knowledge regarding objects' relationships is conveniently compressed in a graph form, and the representation is robot and workspace independent, and easy for users to understand.

A representative example is presented by Wang et al. [3], where they utilize graphs and LfD to learn assembly tasks. They propose a graph based representation of semantics between objects called an *assembly graph* (AG). In the AG each node is a part and each edge is an assembly relation between the respective parts. In the demonstration phase a human teacher then shows how each part can be connected, encoding the necessary information into the AG. Later on, in the execution phase, the robot exploits the information in the AG and replicates a learned task. In this thesis, however, we represent some of the characteristics of the object – the "bumps" in the Lego – as nodes in the graph, as opposed to the whole object. This enables us to use the inexact graph matching techniques to compare different graph structures to one another.

Other examples of using an LfD framework in assembly tasks include [4], where peg-in-hole tasks are learned from a human demonstrator, and [53], in which the LfD framework is utilized in robotic small part assembly. In [4], the task is learned via kinesthetic teaching. The robot learns the trajectories and velocities, as well as the force and torque profiles, that lead to successful execution of the assembly task. Furthermore, the poses of the assembly parts are visually estimated before replicating the task. In [53], the robot observes as a human teacher connects a mobile phone cover to a mobile phone. The robot learns to switch between position, force, and hybrid control in different parts of the task to accomplish the assembly.

2.3 Inexact Graph Matching

As was mentioned earlier, our method attempts to reproduce an observed assembly task even if the robot is given an incomplete set of assembly parts. An incomplete set means that some of the parts that were observed in the demonstration have been changed into parts that were not observed in the demonstration[4]. Therefore, when the robot replicates the observed operations, it must be able to

1. Go through the set of available assembly parts (graphs) and find the ones which are required for the assembly operation, or
2. find and use the most similar assembly parts (graphs) for replication, if identical parts are not available.

We use graphs to model the structure and relations of the assembly parts (the abstraction of the assembly parts into graphs is discussed in detail in Section 3.1). Generally, a graph G is composed of *vertices* $\mathcal{V}$ (also known as *nodes* or *points*) and *edges* $\mathcal{E}$ (also known as *links* or *arcs*), and the graph is defined as an ordered pair $G = (\mathcal{V}, \mathcal{E})$. The links in a graph may be undirected or directed, and unweighted or weighted. An undirected edge signifies that the links have no direction, that is, a link (v_1, v_2) between nodes v_1 and v_2 is identical to the link (v_2, v_1). A directed link, on the other hand, does not have this symmetry, and typically $(v_1, v_2) \neq (v_2, v_1)$.

[4]Note that robot does recognize these parts — the robot knows what they are, and how they can be connected to other Legos.

Furthermore, both undirected and directed links may have weights. The weights depict a quantity between the nodes, such as a cost or a distance. Additionally, two nodes may have *multiple edges* between them, or a single node may have a *loop*, that is, a link that connects to itself. Nonetheless, we use *simple graphs* to model the subassembly parts themselves, as well as their relations to other parts. Simple graphs are undirected graphs without multiple edges or loops. Also, a graph is said to be *connected*, if there is a path from any given node to any other node in the graph. In other words, by traversing the links of a node and its successive neighbors, it is possible to reach any other node in a connected graph. Most of the graphs we examine in this thesis are connected, but not all.

Searching for similar graphs is a well-known problem in many fields, such as computer vision and image processing, and in general all fields that consider, for example, social, biological and chemical networks [54, 55]. If the graphs, which we are comparing, are of the same size we talk about *exact graph matching*. If, on the other hand, the graphs are of different sizes, the term is *inexact graph matching*. Thus, in exact graph matching both graphs have the same number of nodes, and in inexact graph matching the sizes of the graphs can be unequal.

Many similarity measures have been proposed for graph matching, and these techniques can be generally divided into three categories [55]: Those pertaining to *edit distance and graph isomorphism*, *feature extraction and spectral methods*, and *iterative methods*.

Edit Distance and Graph Isomorphism. Two graphs are considered similar if they are isomorphic, or one is isomorphic to a subgraph of the other, or if they have large isomorphic subgraphs. Isomorphic graphs contain an equal number of nodes connected in the same way. Formally, two graphs G and H with nodes $V_n = \{v_1, v_2, \ldots, v_n\}$ are isomorphic if there is a permutation p of V_n such that an edge (v_i, v_j), where $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, n\}$, is in the edge set $\mathcal{E}(G)$ if and only if the edge $(p(v_i), p(v_j))$ is in the edge set $\mathcal{E}(H)$. However, a problem with this approach is that the computational complexity of recognizing graph isomorphism is known to exist in the NP-complete class. It is still unclear whether an algorithm that solves the problem in polynomial time exists, but in special cases, such as *planar graphs* or *trees*, polynomial-time solutions have been found [56, 57].

Graph edit distance is a generalization of the graph isomorphism problem. In graph edit distance the objective is to transform one graph into the other by executing a sequence of operations. These operations can be additions, deletions, or substitutions of nodes or links, or reversions of directed links. Each operation is associated with a cost, and the similarity of two graphs is measured by the minimum cost of operations required to transform one to another. However, graph edit distance is an NP-hard problem [58].

Feature extraction and spectral methods. Feature extraction methods (see, for example, [59] and [60]) are based on the assumption that similar graphs share certain properties, such as degree distribution, clustering coefficient, eigenvalues, and so on. By comparing these features we can measure the similarity between two graphs. Spectral methods, in particular, use the eigenvectors and eigenvalues of the adjacency or *Laplacian matrices* to measure the similarity of graphs. In general, these feature extraction and spectral methods are powerful and scale well as we only need to extract chosen, typically easily computable, statistics from the graphs.

Iterative methods. In this class of methods the idea is that nodes in separate graphs are similar if the nodes' neighborhoods are similar. Thus, the nodes of the graphs are iterated through, and during each iteration the nodes exchange similarity scores until convergence is achieved. Some successful methods that belong to this category include the *similarity flooding algorithm* by Melnik et al. [61], and *SimRank* by Jeh and Widom [62].

In this thesis we use spectral methods to measure the similarity of graphs. The methods based on graph isomorphism and graph edit distance are difficult to implement, and do not scale well to large assemblies [58]. The iterative methods may not be applicable to our graphs because we typically have multiple similar nodes that represent the same Lego brick, hence the neighborhoods of all these nodes are similar.

2.3.1 Spectral Methods

Spectral methods utilize the following observation: Eigenvalues of a graph's adjacency matrix are invariant with respect to node permutations [60]. Therefore, if two graphs are isomorphic, their eigenvalues will be identical. The set of eigenvalues is called the spectrum of the graph, and in addition to the adjacency matrix, the spectrum can also be computed, for instance, from the Laplacian matrix of the graph. The interest in spectral methods is furthered by the fact that the computation of eigenvalues and eigenvectors is a well known problem and solvable in polynomial time.

In 1988 Umeyama [63] introduced a spectral method for comparing two graphs. This pioneering work was based on spectral graph theory and the fact that the eigenvectors and eigenvalues of an adjacency matrix — or a Laplacian matrix — characterise the structural properties of graphs. Thus, we can compare the structure of two graphs by comparing their eigenvectors and/or eigenvalues. In the method of Umeyama the comparison of graphs G and H is performed by transforming the graph G into the space of graph H. This creates an approximation G' of graph G in the space of the graph H. The difference between graphs G and H is then approximated by the difference between G' and H.

Caelli and Kosinov [64] proposed another approach, in which instead of transforming graph G into the space of graph H, both graphs are projected into their own

eigenspaces. As this method was suggested for inexact graph matching, the eigenspace projections must be renormalized by projecting them into a unit hypersphere defining the eigenspaces. Two graphs are then similar if their rescaled eigenvectors have similar angles and their eigenvalues are similar. However, as Lee and Duin [65] points out, this comparison does not actually satisfy the original intent of Caelli and Kosinov in executing the comparison in a common eigenspace of the graphs. Also, in the method of Umeyama the comparison is not symmetric, since the graphs are compared in the original space of one of the graphs. Lee and Duin proposes a symmetric similarity measure for inexact graph comparison in [65]. In order to do a fair comparison of two graphs, they are mapped into the same eigenspace and then compared in this unique eigenspace.

We present two different spectral methods for measuring the (dis)similarity between two graphs. In the first we compute the eigenvalues of the graphs' Laplacian matrices, and the dissimilarity measure is the Euclidean distance between these eigenvalues. The greater the distance, the more dissimilar the graphs are. The distance between eigenvalues is a commonly used dissimilarity measure for graph matching (see, for example, [59, 66]). The second measure is the symmetric dissimilarity measure *Joint Eigendecomposition* (JoEig) proposed by Lee and Duin [65]. In the experiments we compare the performance of these two dissimilarity measures: The objective is to build an assembly product that is as similar as possible to the one that was observed during learning. Note that, throughout this thesis, we will be talking about maximizing similarity using a similarity measure, or minimizing dissimilarity using a dissimilarity measure. These expressions are equivalent, and the dissimilarity and similarity measures refer to these Laplacian and JoEig dissimilarity measures.

Eigenvalues of the Laplacian Matrix. In this thesis we consider only undirected, unweighted graphs $G = (\mathcal{E}, \mathcal{V})$, where the node set is $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ and the edge set is $\mathcal{E} = \{e_1, e_2, \ldots, e_m\} \subset \mathcal{V} \times \mathcal{V}$. An adjacency matrix of such a graph is defined as

$$\boldsymbol{A}_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

Wilson and Zhu [59] argues that a Laplacian matrix is superior against an adjacency matrix when used as a similarity measure. A Laplacian matrix $\boldsymbol{L}$ for a graph without loops and multiple edges is computed as

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}, \tag{2.2}$$

where $\boldsymbol{A}$ is the adjacency matrix and $\boldsymbol{D}$ is a diagonal *degree matrix*, whose diagonal elements are given by the node degrees $\boldsymbol{D}_{ii} = p_i$, p_i being the number of links attached to the node. The degree matrix is computed by simply summing over the rows or columns of an adjacency matrix. An equivalent description of the Laplacian

matrix is

$$\boldsymbol{L}_{ij} = \begin{cases} p_i, & \text{if } i = j, \\ -1, & \text{if } (v_i, v_j) \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

Additionally, the *symmetric normalized Laplacian matrix* $\boldsymbol{\mathcal{L}}$ is defined for a general graph without isolated nodes as

$$\boldsymbol{\mathcal{L}} = \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}, \tag{2.4}$$

where $\boldsymbol{I}$ is the identity matrix. The symmetric normalized Laplacian matrix can also be described as

$$\boldsymbol{\mathcal{L}}_{ij} = \begin{cases} 1, & \text{if } i = j \text{ and } p_i \neq 0, \\ -\frac{1}{\sqrt{p_i p_j}}, & \text{if } (v_i, v_j) \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.5}$$

This normalized Laplacian matrix has some desirable properties, such as 1) real and non-negative eigenvalues, 2) eigenvalues are bounded above by 2, and 3) the eigenvalues of $\boldsymbol{\mathcal{L}}$ are related to such invariant properties of the graph as *connectedness*, *algebraic connectivity* and *Cheeger inequality* [59, 67][66, chap. 1].

Let us consider two graphs, G and H with normalized Laplacian matrices $\boldsymbol{\mathcal{L}}_G$ and $\boldsymbol{\mathcal{L}}_H$. The eigendecomposition of both Laplacian matrices is performed as

$$\begin{aligned} \boldsymbol{\mathcal{L}}_G &= \boldsymbol{V}_G \boldsymbol{D}_G \boldsymbol{V}_G^T \\ \boldsymbol{\mathcal{L}}_H &= \boldsymbol{V}_H \boldsymbol{D}_H \boldsymbol{V}_H^T, \end{aligned} \tag{2.6}$$

where $\boldsymbol{V}_G$ and $\boldsymbol{V}_H$ are orthonormal eigenvector matrices[5], and $\boldsymbol{D}_G$ and $\boldsymbol{D}_H$ are diagonal matrices containing the eigenvalues in ascending order. Now we can compute the dissimilarity $\delta_{Laplacian}$ between graphs G and H as

$$\begin{aligned} \delta_{Laplacian}(G, H) &= \|diag(\boldsymbol{D}_G) - diag(\boldsymbol{D}_H)\|_2 \\ &= \sqrt{\sum_{i=0}^{K} (\boldsymbol{d}_{G,i} - \boldsymbol{d}_{H,i})^2} \\ &= \sqrt{(\boldsymbol{d}_{G,1} - \boldsymbol{d}_{H,1})^2 + (\boldsymbol{d}_{G,2} - \boldsymbol{d}_{H,2})^2 + \cdots + (\boldsymbol{d}_{G,K} - \boldsymbol{d}_{H,K})^2}. \end{aligned} \tag{2.7}$$

In the above, $diag(\boldsymbol{D}_G)$ is a vector containing the diagonal elements of $\boldsymbol{D}_G$, that is, the eigenvalues. Similarly, $diag(\boldsymbol{D}_H)$ is a vector containing the diagonal elements of $\boldsymbol{D}_H$, and $\boldsymbol{d}_{G,i}$ and $\boldsymbol{d}_{H,i}$ are the i^{th} eigenvalues of G and H, respectively. Thus, the dissimilarity between graphs G and H is measured as the Euclidean distance between the eigenvalues of their symmetric normalized Laplacian matrices.

Generally there is a different number of nodes in the graphs G and H, which means that there is a different number of eigenvalues in both graphs — the number of

[5]The orthonormality results from the fact that the symmetric normalized Laplacian matrix of an undirected graph is symmetric.

eigenvalues is equal to the number of nodes. Thus, in equation (2.7) there are three straightforward options as how many eigenvalues we use, that is, what we choose as the value of K. Firstly, we may choose K to be the number of eigenvalues in the larger graph, which means that we have to pad the vector containing the eigenvalues of the smaller graph with zeros to make the dimensions match. Second choice is to choose K to be the number of eigenvalues in the smaller graph, in which case we must ignore the smallest eigenvalues of the larger graph. Thirdly, we can choose to use only the K largest eigenvalues, such that K is smaller than the number of eigenvalues in either graph. In the experiments we chose the third alternative, and use only $K = M - 2$ largest eigenvalues, where M is the size of the smaller graph. The smallest eigenvalue is ignored, because it is always zero when the graph is connected — which typically is the case in our experiments. The second smallest eigenvalue was found to be significantly smaller than the other eigenvalues, and was thus ignored.

Joint Eigendecomposition. The second method which we use to measure the similarity of graphs is the JoEig [65]. In this method both graphs are projected into a joint eigenspace, which is expanded by both sets of eigenvectors. Lee and Duin [65] present the derivation of the measure as follows. We begin by considering two graphs G and H, and perform the eigendecomposition of their adjacency matrices $\boldsymbol{A}_G$ and $\boldsymbol{A}_H$

$$\begin{aligned} \boldsymbol{A}_G &= \boldsymbol{V}_G \boldsymbol{D}_G \boldsymbol{V}_G^T \\ \boldsymbol{A}_H &= \boldsymbol{V}_H \boldsymbol{D}_H \boldsymbol{V}_H^T. \end{aligned} \tag{2.8}$$

The orthonormal properties of $\boldsymbol{V}_G$ and $\boldsymbol{V}_H$ are exploited to rewrite the above equation as

$$\begin{aligned} \boldsymbol{D}_G &= \boldsymbol{V}_G^T \boldsymbol{A}_G \boldsymbol{V}_G \\ \boldsymbol{D}_H &= \boldsymbol{V}_H^T \boldsymbol{A}_H \boldsymbol{V}_H. \end{aligned} \tag{2.9}$$

Let us assume for now that the graphs G and H are isomorphic. In this case their eigenvalues are identical $\boldsymbol{D}_G = \boldsymbol{D}_H$, and therefore

$$\boldsymbol{V}_G^T \boldsymbol{A}_G \boldsymbol{V}_G = \boldsymbol{V}_H^T \boldsymbol{A}_H \boldsymbol{V}_H. \tag{2.10}$$

This can be further modified into

$$\boldsymbol{A}_G \boldsymbol{V}_G \boldsymbol{V}_H^T = \boldsymbol{V}_G \boldsymbol{V}_H^T \boldsymbol{A}_H, \tag{2.11}$$

where $\boldsymbol{V}_G \boldsymbol{V}_H^T$ is the joint projection vector for both graphs G and H. By introducing equation (2.8) into equation (2.11), and removing terms that correspond to identity, we get

$$\boldsymbol{V}_G \boldsymbol{D}_G \boldsymbol{V}_H^T = \boldsymbol{V}_G \boldsymbol{D}_H \boldsymbol{V}_H^T, \tag{2.12}$$

and, therefore, the dissimilarity measure δ_{JoEig} as proposed by Lee and Duin is

$$\delta_{JoEig}(G, H) = \| \boldsymbol{V}_G \boldsymbol{D}_G \boldsymbol{V}_H^T - \boldsymbol{V}_G \boldsymbol{D}_H \boldsymbol{V}_H^T \|_2^2. \tag{2.13}$$

Note that the products $\boldsymbol{V}_G\boldsymbol{D}_G\boldsymbol{V}_H^T$ and $\boldsymbol{V}_G\boldsymbol{D}_H\boldsymbol{V}_H^T$ in equation (2.13) are matrices, and that the 2-norm is taken "entrywise." Hence, the dissimilarity measure is

$$\delta_{JoEig}(G,H) = \|\boldsymbol{W}\|_2^2 = \left(\sum_{i=1}^{m}\sum_{j=1}^{n}(w_{ij})^2\right), \tag{2.14}$$

where $\boldsymbol{W} = \boldsymbol{V}_G\boldsymbol{D}_G\boldsymbol{V}_H^T - \boldsymbol{V}_G\boldsymbol{D}_H\boldsymbol{V}_H^T$, m and n depict the size of the matrix $\boldsymbol{W}$, and w_{ij} is an element of the matrix $\boldsymbol{W}$ with row index i and column index j.

However, as the sizes of the graphs typically are different, the matrix products in equations (2.10) – (2.13) may not be defined. Hence the same problem with dimensionality befalls this method as well, and the number of eigenvalues K has to be chosen as was discussed earlier. Lee and Duin [65] found in their experiments that the best results were generally produced when the number of eigenvalues was chosen to be either the size of the smaller graph, or the size of the smaller graph minus a small number[6]. Unlike in the Laplacian dissimilarity measure, now all the eigenvalues are more similar in magnitude. Thus, we chose to use $K = M$ eigenvalues — where M is the size of the smaller graph — to retain as much information as possible from the smaller graph.

2.3.2 Inexact Graph Matching in Assembly Tasks

Inexact graph matching has been rarely used to compare assembly products or subassembly parts (for example [6], which we discuss below). One reason might be, that in the previously discussed publications the graphs have been mainly concerned with operations such as *"At which point can we efficiently and conveniently join part A to part B"* (for example, the AND/OR graphs of de Mello and Sanderson [5]). However, we are more concerned with finding a graph that is as similar as possible to a given query graph.

In [7], Bauckhage presented a visual assembly recognition system. The assembly products were created using a wooden Baufix construction kit. The Baufix parts are rather like Legos, but they have more complex shapes and are attached in a different manner. Bauckhage used a combined neural and semantic net approach, presented in [68], to recognize assembly products from stereo images. The recognizer in [68] identifies the elementary Baufix parts from the images, and uses the relative positions of the parts to create a *mating feature graph* of the assembly product. The nodes in this graph depict segments of the elementary parts, and the structural relations between the segments define the links. For example, a link may have been labelled as *"ADJ"*, which means that nodes belong to the same elementary part. These mating feature graphs are then used to recognize an encountered assembly product: A graph extracted from a stereo image is compared to other graphs in a database. The comparison is executed using graph edit distance. Furthermore, a

[6]Five in their experiments, but they had larger graphs.

mating feature graph and its corresponding stereo image can be used to reconstruct an observed assembly product as a CAD model.

As mentioned in Section 2.1, Ferch et al. [6] use a graph matching technique to compare the similarity of two subassembly products. Ferch et al. model the assembly products as aggregate models. The model is composed of a set of parts that compose the product, a set of liaisons, which describe how the parts are connected, and a known position in the world coordinates. Ferch et al. use four different types of parts in their assembly products: screws, ledges, blocks, and nuts. In the experiments, three Puma 260 robots manipulate these parts by using plugging or screwing operations. Similarly to our method, the parts can be connected to each other only through predefined points. The method by Ferch et al. uses graph edit distance to assess the similarity between two subassembly products. Thus, their method attempts to find approximately similar assembly structures, when exact structures are not possible. The aggregate models are represented as tree structures, and therefore the graph edit distance is the cost of transforming one tree structure to another. However, their models of the assembly products allow multiple tree representations for a single product. Thus, to compare different subassembly products, they need to first make the tree structures unique. In the experiments the system is given an aggregate model, which it attempts to create using the available parts. The method by Ferch et al. uses Q-learning to figure out the required actions to create an assembly product that corresponds to the given model.

Sundar et al. [69] propose a method for finding similar 3D objects based on graph matching techniques. The objects are abstracted into graphs as follows: First, a volumetric or a polygonal (3D) model of an object is obtained. Then, a set of skeletal nodes is extracted from the model, and the nodes are connected to form a graph. Finally, the graph is indexed into a database. The topological information of an object is thus encoded in the form of a graph. Additionally, a *topological signature vector* is computed for each node. The signature vector is a low-dimensional descriptor that captures local structural properties of the graph. Afterwards, when one desires to classify a novel object, or find similar objects, he can query the database using the shape information of the novel object. The matching of the queried graph and the graphs in the database is formulated as a problem of finding a subset of nodes with maximum cardinality and minimum weight from a bipartite graph. This bipartite graph is formed by connecting each node of the queried graph into all nodes of a graph from the database. The weight of these links represent the distance between the two nodes' topological signature vectors. The subset of nodes in the bipartite graph which provides a minimum sum of weights, and a maximum cardinality number, represents the object that resembles most the queried graph. Note that the best matching graph might contain the queried graph only as a subgraph.

The approaches described above differ from our method in how the graphs are defined and used. The way Bauckhage [7] uses graphs is quite similar to our graph representation: The nodes represent object parts which can be connected to other objects. However, the links between the nodes are defined differently. Instead of

labelling the links, all links in our system are similar. This allows us to use spectral inexact graph matching to compare assembly products. Bauckhage uses graph edit distance to compare graphs, which means that he has to manually define how much the different operations (adding or removing a node or a link, or changing the label of a link) cost. Different costs change the measured similarity between two graphs. In our chosen spectral dissimilarity measures the only parameter that requires tuning is the number of eigenvalues we want to use. Moreover, Bauckhage uses inexact graph matching to check whether an assembly product has been encountered before. We use inexact graph matching to check whether the required assembly parts are available, or to create a structure that resembles an observed structure. Also, Bauckhage forms the graphs from the images of a complete assembly product, which is prone to errors due to occlusion: An occluded subassembly part will not be recognized, and thus will not be a part of the graph. In our system the assembly product is constructed in steps, which reduces the problems caused by occlusion.

In the assembly method by Ferch et al. [6], the assembly parts are ultimately represented as tree structures. The similarity between the tree structures is measured using graph edit distance. In our method, however, we use the adjacency or Laplacian matrix representations of the graphs to find similar structures. Also, as noted earlier in this section, the graph edit distance is not an ideal method for comparing large structures as it is an NP-hard problem. The structures used in this thesis are not particularly large, but nevertheless, scalability is a desired property for a general assembly skill. Furthermore, their model representation allows multiple tree structures for the same assembly product, which is not an issue in our method. Additionally, in [6], there is no learning of an assembly task involved. Instead, the robots are just given the model of the structure they should assemble.

Lastly, in the method of Sundar et al. [69], the objects are described as graphs in a slightly similar manner to our method. The structure of a single object is modelled as a graph, and the nodes form a topological shape of the object. However, in our representation of the assembly parts, the nodes have a functional role — they are places where another Lego may be connected to. In the method by Sundar et al., the nodes do not have such a role. Instead, the nodes contain descriptors of the graph's local geometry. These descriptors are used to compare different graphs to each other, whereas we compare graphs to each other using their spectra. As the reader might have noticed, Sundar et al. do not use the method for assembly purposes, although it could be applied for such intentions. Thus, their work cannot be compared to our approach as an assembly method.

3 Teaching and Reproducing Assembly Tasks

The objective of this thesis is to formulate and examine an assembly method in the LfD framework using a graph representation of the assembly parts and their spatial relations. Additionally, the method uses inexact graph matching to replicate observed tasks if it is provided with imperfect knowledge of the assembly operations. The method is used to observe an assembly task demonstrated by a human teacher, and subsequently to replicate the task. Each task is comprised of a sequence of assembly operations. During an assembly operation, the teacher manipulates an assembly part and moves it from one location to another. The viability of our approach is examined in Section 4, where we evaluate our method through a series of experiments.

Each assembly task is learned and replicated in two separate phases. In the *learning phase* a human teacher demonstrates the assembly task in front of a Kinect sensor. Our system then processes the acquired RGB-D data to learn the assembly operations. In the *replication phase* a robot then attempts to replicate the observed operation. Note that, throughout this thesis, we will generally refer to our system as a *robot*. For example, we might say that a robot observes, learns or replicates an assembly task, but we mean that our system observes, learns or replicates the task. There is no real world manipulator involved in the experiments in this thesis — instead, all the replications of observed assembly tasks are run as simulations. However, such a manipulator could be added into the system.

Figure 3.1 displays a general overview of how the learning and replication phases work. When a robot observes an assembly task, it must first identify the assembly parts which are used in the task. Then, it monitors as the human teacher performs assembly operations using the parts. Each operation is saved into a database: Ultimately, this database contains all operations that are required to complete the observed task. Afterwards, the robot utilizes the learned database in the replication phase. When the robot replicates an observed task, it must again begin by identifying the assembly parts which are provided to it. Then, it reads the assembly operations from the database, and attempts to reproduce them. Furthermore, the reproduced operation depends also on what kind of assembly parts are provided to the robot. If the robot is supplied with the same set of assembly parts as it observed during the demonstrations, it will simply replicate the learned operations. However, if the provided parts do not match the observed ones, the robot must perform an operation that results in an assembly product that resembles the observed one.

In this section we will discuss in detail how the proposed assembly method functions. First, Section 3.1 explains thoroughly how the assembly parts and products are represented as graphs. Then, in Section 3.2 we describe how the robot observes and learns an assembly operation: How the assembly parts are recognized, how their paths are tracked during manipulation, and how these operations are saved into the database. Finally, Section 3.3 specifies how the robot exploits the database to replicate the observed assembly operation.

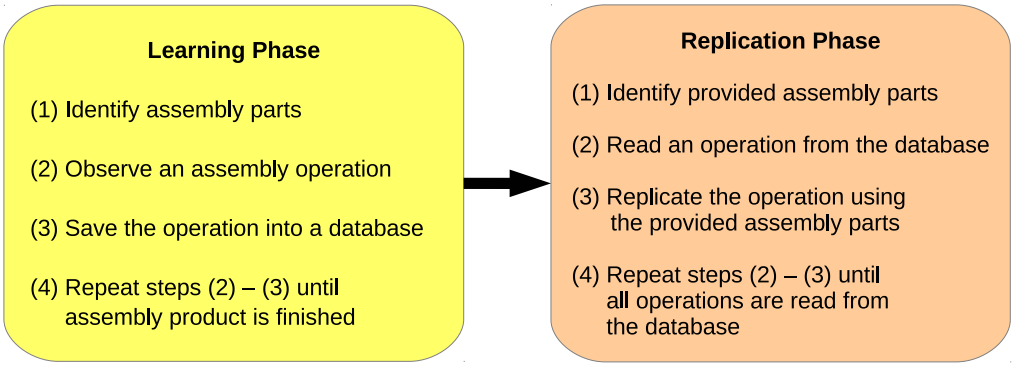


Figure 3.1: *A schematic of our proposed assembly method. The method is divided into two phases: In the learning phase the assembly task is learned, while in the replication phase the assembly task is reproduced. Note that we will be using this same color coding throughout this thesis. The yellow background refers to actions that occur in the learning phase, while the reddish background refers to actions that occur in the replication phase.*

3.1 Abstraction of Object Compositions into Graphs

As previously discussed in Section 2.1, graph structures provide a compact and intuitive representation of objects' topological characteristics, and their relationships. Consequently, it is not uncommon to see graphs used in assembly tasks for modelling the structure of assembly products and planning action sequences. Additionally, the utilization of graphs enables us to use techniques such as inexact graph matching to compare different assemblies to one another.

Our graph representation is as follows. The nodes of the graph correspond to locations of potential attachment between subassembly parts. Hence, links between the nodes represent the coalescence of objects. As we use unweighted links, each link between objects is assumed to be equally strong. Therefore, in principle, two objects are attached to each other with a linkage, whose strength is proportional to the number of links between the nodes that are part of those objects. We could exploit this information to, for example, uncover the weakest link between assembly parts. However, this information is not used in the experiments.

We chose to use Lego Quatros in this thesis for a number of reasons: They are simple to manipulate, we can construct a variety of shapes using them, and they are large enough for the Kinect to see at a distance. More specifically, Quatro bricks are four times the length, width, and height of regular Lego bricks and 64 times the size in volume. Furthermore, they are two times the length, width, and height of Duplo bricks and 8 times the size in volume. See Figure 3.2 for a visual comparison of a regular Lego, a Duplo and a Quatro. In addition, Quatros are compatible with Duplos, and Duplos are compatible with regular Legos.

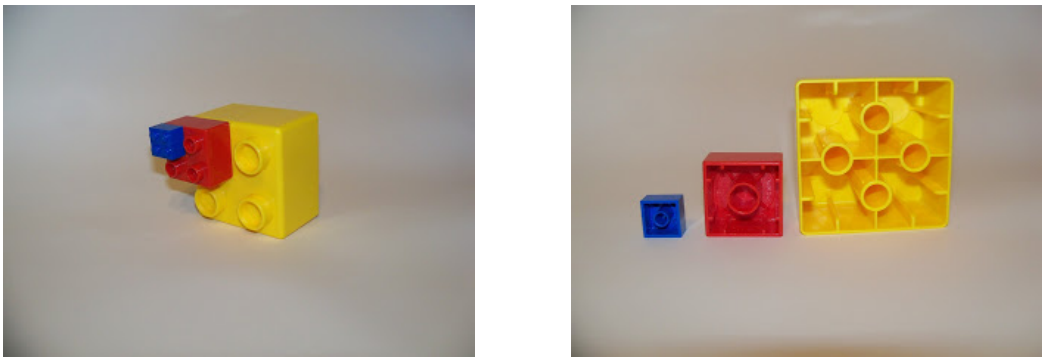


Figure 3.2: *Comparison of a regular Lego, a Duplo and a Quatro. Lego Quatro is 64 times the size of a regular Lego in volume, and 8 times the size of a Lego Duplo. Images from* `http://www.dagsbricks.com/2013/07/lego-techniques-using-quatro.html`.

We use only two kinds of Lego Quatros in the experiments, a 2x2 brick and a 2x4 brick — the color of the Legos is irrelevant. The Quatro bricks are abstracted into a graph such that the nodes correspond to the "bumps" in the brick. In other words, each node in the graph corresponds to a certain position in the Lego which can be connected to another Lego brick. The links represent which nodes are connected. A node in one Lego brick is always connected to every other node inside the same Lego block. Then, once an assembly operation is performed, a node may be connected to one, and only one, node in another Lego brick. In a standard 2x2 Lego Quatro there are 4 regions on the top and 4 regions on the bottom where another brick can be connected. We define each of these regions to be one node. Hence, there are a total of 8 nodes in a 2x2 Quatro brick, as Figure 3.3 shows. Similarly, in a 2x4 Quatro brick there are 8 regions on top and 8 regions on bottom, thus totalling 16 nodes in this brick. With the chosen representation we can directly observe whether a node is occupied[7], and whether it is located on the top or on the bottom of the Lego. These two pieces of information are extremely important in inexact replication, where the method systematically goes through configurations of available Legos to find one that is similar to the learned structure.

This abstraction is generalizable to many kinds of assembly parts and products. Essentially, one has to model the object as a set of nodes, and train the robot to perform operations on these nodes. In this thesis we have used simple pick-and-place operations, but any other type of operations — for instance, screwing operations — could also be used. However, if other types of operations are added to the system, some new functionalities are required to the software — for example, to define how one part is screwed to another.

Throughout this thesis, we will generally refer to a structure that consists of multiple Legos as a *block (of Legos)*, an *assembly product* or a *structure*. If we refer to a single Lego brick, we will simply call it a *(Lego) brick*.

[7]We define a node to be occupied if it is connected to another subassembly part, or in this case, another Lego. Likewise, an unoccupied node is not connected to another subassembly part.

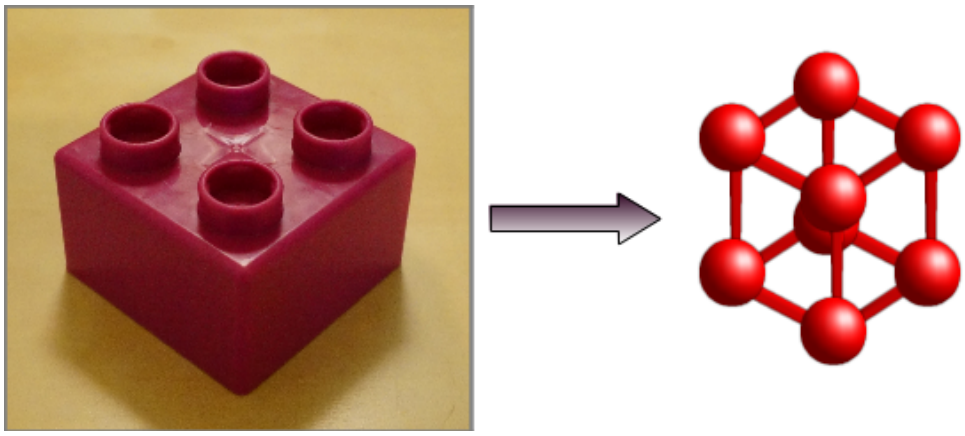


Figure 3.3: *The abstraction of a 2x2 Lego Quatro into a graph. The nodes correspond to the bumps in the Lego. Each node is connected to every other node in the same Lego brick, but for clarity only the links that outline the structure are shown here.*

3.2 Learning Symbolic Assembly Tasks

When a teacher demonstrates an assembly task to the robot, he first places all the necessary Lego bricks in front of the Kinect on a tabletop. Unnecessary Lego bricks can be placed on the table as well, but they will be ignored if they are not used in the task. Our method then identifies the available bricks, and estimates their poses. This recognition step occurs only in the beginning, and later on the locations of the Legos are updated during the tracking process.

The assembly tasks are learned as a sequence of assembly operations. After the Legos have been initially recognized, the teacher informs our program which Lego brick he intends to move. The program then tracks the designated brick as the teacher places it to a novel location, either besides or on top of another Lego.[8] In the course of the assembly task, the teacher can also place a larger block of Legos on top of another structure: The tracking process is not limited to single bricks. Each assembly operation is then encoded into a database containing all the operations that are required to execute an assembly task successfully.

First, Section 3.2.1 presents the methods that are used in the recognition and tracking processes. These recognition and tracking processes are based on earlier work (especially [70–72]). Then, Section 3.2.2 introduces our database of assembly operations, which contains the learned assembly operations. Lastly, in Section 3.2.3, the whole teaching process is presented thoroughly.

[8]To be exact, the designated brick may also be placed such that it remains hanging from the other structure. This is, however, quite a challenging assembly operation, since the brick may fall off. Thus, we do not use it in the demonstrations.

3.2.1 Recognition and Tracking of a Lego Quatro

In order to learn an assembly task, the assembly parts must be identified correctly and their poses estimated precisely. As we use two different types of Quatro bricks, we must be able to discriminate reliably which types of Legos are present in front of the Kinect. We also need an accurate estimation of the bricks' poses with relation to the camera's coordinate system. This enables us to compute the positions of the Legos with relation to each other before, during, and after the assembly. Once the available Lego bricks have been recognized, the teacher conducts a series of assembly operations. In each assembly operation, the teacher places a block of Legos either besides or on top of another block of Legos. Therefore, we need a tracking method to monitor the movements of the Lego blocks. We will first discuss how a Lego Quatro is recognized, and then how the tracking method functions.

Recognition of a Lego Quatro. The recognition of an object generally takes place using either a local descriptor (for example, *signature of histograms of orientations*, SHOT [73], or *point feature histograms*, PFH [74]), or a global descriptor (for example, *viewpoint feature histograms*, VFH [75], or *clustered viewpoint feature histograms*, CVFH [76]). Essentially, local descriptors utilize local features around distinct key points. This requires that the object has distinguishable characteristics, such as an unique shape or texture. In the recognition of an unknown point cloud the key points are extracted, clustered, and matched to known compositions of the key points. If a match is found, the point cloud is recognized. Global descriptors, on the other hand, operate at the level of an entire object, and hence use features that are computed from a larger surface area. Since the global descriptors use the whole objects, a preprocessing step is required to segment the objects from the background scene of a given point cloud. This step is extremely important, as a failed segmentation generally implies failed recognition. Typically, the objects which are used in the recognition must fulfill certain restrictions: They have to be rigid, and cannot be reflective or transparent.

We use a semi-global descriptor known as *oriented, unique, and repeatable clustered viewpoint feature histogram* (OUR-CVFH, [70]) to recognize and estimate the pose of the assembly parts. We chose this descriptor because the Lego Quatros are reasonably small, textureless, and symmetrical. A global descriptor takes advantage of the whole shape of the Quatro instead of focusing on rather similar key points. Moreover, the OUR-CVFH descriptor should cope with partially occluded objects and noise from the Kinect sensor [70]. Also, the descriptor works with synthetic models, which makes the training of the recognizer straightforward: There is no need to create a descriptor database by taking images of an object from multiple different viewpoints — instead, the viewpoints can be virtually generated from the synthetic model.

The OUR-CVFH descriptor is built upon the CVFH descriptor and *semi-global unique reference frames* (SGURF, [70]). The SGURFs are smooth regions that encode the geometrical properties of an object's surface. Additionally, the SGURFs are used to

unambiguously define the 6DOF pose of a detected surface. For a detailed account of how the OUR-CVFH descriptor and SGURF works, see [70, 76].

In this thesis we use the KukaVision software[9], which implements OUR-CVFH descriptors for recognizing objects. The software must be taught to recognize the designated objects, for which it requires partial views of the objects' 3D models. We will discuss the teaching process and the software in more detail in Section 3.2.3.

Tracking of a Lego Quatro. We use the *adaptive particle filter tracker* to track the movements of the assembly parts during the assembly operations. The adaptive particle filter tracker is implemented in the *Point Cloud Library* (PCL, [77]). However, the tracking method yields only a rough estimation of the new location and pose of the manipulated Legos. Hence, once a Lego block has been moved, we employ the *iterative closest point* (ICP, [78]; also implemented in the PCL) algorithm to accurately estimate the new pose of the block. It is essential that the locations are estimated precisely: As we update the graph representations of the Legos, we consider two nodes connected only if the distance between them is smaller than 12 millimetres. Consequently, a poorly estimated position results in missing links in the graph structure and an erroneous assembly operation.

The PCL's tracking method uses a *particle filter* algorithm (see [71, 79] for a detailed account of particle filters) to implement 6DOF pose tracking, and is optimized to run in real-time. In general, the algorithm works by iterating these stages at each time step t:

1. Use previous particles' (at time step $t-1$) information about the object's position and rotation to predict the pose at the current time step.
2. Calculate weights for the particles by using likelihood functions for, for instance, normals and color of the object that is tracked.
3. Compare real point cloud data from the depth sensor with the predicted particles, and resample.

Additionally, the PCL's adaptive particle filter tracker utilizes *adaptive KLD-sampling* [72], which adapts the number of the particles on-the-fly according to the uncertainty of the pose.

3.2.2 Database of Assembly Operations

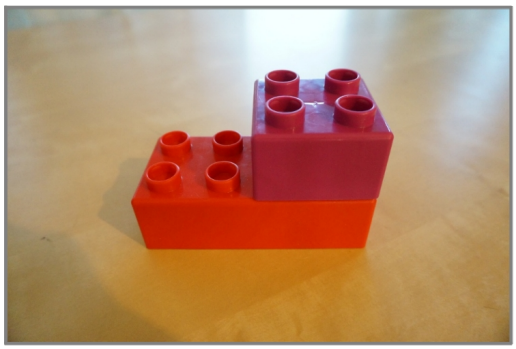
Each of the assembly operations, that is, a movement of a Lego block relative to another Lego block, is encoded into a database. This database is later exploited in order to replicate an observed assembly task.

An individual assembly operation is encoded into the database using four pieces of data: three adjacency matrices, and one transformation matrix. The first two

[9] Available at `https://github.com/lucatlp87/KukaVision`.

adjacency matrices are the adjacency matrices of those blocks which are joined together in the operation. These adjacency matrices are used in the replications to find the initial blocks, that are required to successfully repeat the assembly operation. The third adjacency matrix is the adjacency matrix of the resulting assembly product. This adjacency matrix indicates what is the desired configuration of Legos after each operation in the assembly task. This will be used in the inexact replications, when the required blocks are not available, and thus the assembly operation cannot be replicated exactly. In the inexact replications we attempt to find a configuration of the currently available Legos, that is as similar as possible to this desired adjacency matrix of the post-operation assembly product. The last piece of data, which is saved into the database, is the transformation matrix between the post-operation blocks. This transformation matrix describes the transformation from the coordinate system of the block that was not moved, to the coordinate system of the block that was moved. In other words, the transformation matrix describes what are the poses of the blocks with relation to each other after the assembly operation. This transformation matrix is used in the exact replications to move a block to a desired position: The assembly product, described by the third adjacency matrix, is produced when the initial blocks are located according to the transformation matrix.

You can find a simple example configuration of two Legos in Figure 3.4. The figure also displays the adjacency matrix that corresponds to this structure. If the robot observed an assembly operation where the purple 2x2 brick is placed on top of the red 2x4 brick, this would be the desired adjacency matrix of the assembly product. Remember that the smaller brick contains 8 nodes that are all connected to one another and the larger brick contains 16 nodes. The upper left corner of the adjacency matrix in Figure 3.4 corresponds to the adjacency matrix of the smaller brick. Likewise, the lower right corner of the adjacency matrix depicts the adjacency matrix of the larger brick. The “1”s outside these two clusters correspond to the links between these Lego bricks.



$$\boldsymbol{A} = \begin{bmatrix}
0&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&0&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&0&1&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&1&0&0&1&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&1&0&1&1&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&1&1&1&1&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&1&1&1&1&1&0&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&0&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&0&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&0&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&0&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&0&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&0&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0
\end{bmatrix}$$

Figure 3.4: *An example of a simple configuration, and its corresponding adjacency matrix. The clusters of “1”s in the upper left corner and lower right corner represent the smaller and the larger brick, respectively. The “1”s outside these clusters represent the links between the bricks.*

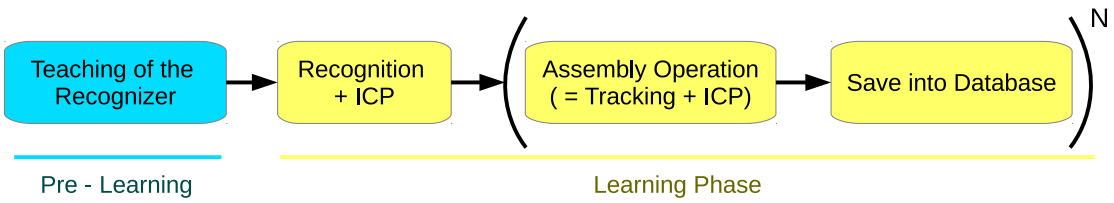


Figure 3.5: *The learning pipeline. This pipeline depicts how an assembly task is learned. Before the teacher can initiate the demonstrations of any assembly tasks, the recognizer must be taught to identify and estimate the type and pose of the objects. Note that this step must be performed only once, and is not part of the assembly tasks, per se. The recognizer is then used in the beginning of a demonstration to identify the objects and estimate their poses. Afterwards, the teacher moves the Legos one by one whilst the system tracks the position of the object. These movements, and the graph structures that are involved, constitute the assembly operations which are saved into a database.*

3.2.3 Learning Pipeline

In this section we present the learning pipeline, which describes how an assembly task is learned by observing an demonstration. As Figure 3.5 shows, the pipeline is divided into distinct segments. In the very beginning, before any assembly tasks can be learned, we must teach the recognizer[10] what kind of objects it has to recognize. Then, we employ the recognizer to identify the types and poses of the objects that lay in front of the Kinect sensor on the tabletop. These are the objects which the human teacher uses in the demonstration. After the objects are recognized, the teacher moves one Lego brick at a time to a new location, either on top of another Lego or besides it. We employ the tracker to monitor the movements of the Legos. When the Lego is moved to the new position, the observed operation is saved into the assembly operation database. This tracking and update of the database step is iterated as long as the teacher wishes, or runs out of disconnected Lego blocks.

Teaching of the Recognizer. As was mentioned earlier, we use the KukaVision software to identify the objects and estimate their poses. In order to teach the software to recognize the Legos, we need 3D models of the objects (see Figure 3.6) and a set of partial views of the models. The models are hand-crafted using a free online tool available at `https://www.tinkercad.com`. We generate the set of partial views using PCL (see Figure 3.7 for an example), and then use them to teach the recognizer. Essentially, the KukaVision computes a set of OUR-CVFH descriptors for each partial view, and later in the recognition phase attempts to match an observed point cloud into these partial views.

[10]From here on, we will call the recognition and tracking processes — or the KukaVision software and PCL's adaptive particle filter — as the recognizer and tracker, respectively.

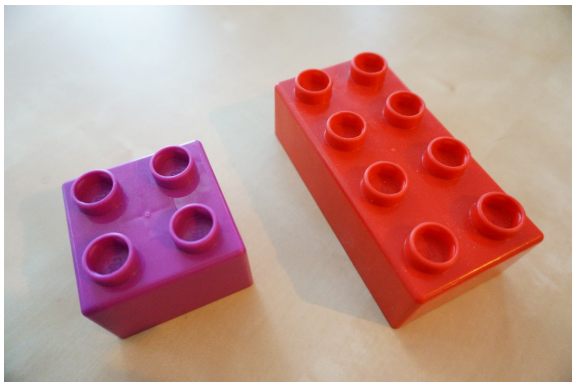
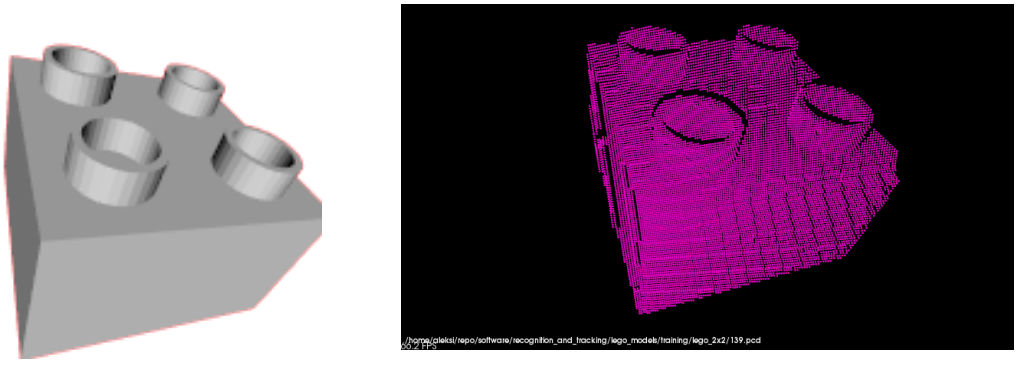


Figure 3.6: *In this thesis we use only two kinds of Legos, a 2x2 brick and a 2x4 brick. The color of the bricks is irrelevant.*

Recognition + ICP. We modified slightly the source code of the KukaVision software. Firstly, the parameters were adapted to better suit the recognition of the Legos. The parameters that control the search of SGURFs in the surface of a point cloud were set high: `curvatureThreshold`, `clusterTolerance`, and `EPSAngleThreshold` were set to 1.0, 0.1 and 3.0, respectively. These high values ensure that the whole surface of the Lego is used to compute the SGURFs and the OUR-CVFH descriptors. We chose to use the whole surfaces to compute the descriptors, because initial tests suggested that then the software provides more accurate estimates of the bricks' poses. Secondly, in a preprocessing step the segmented clusters were smoothed using a *moving least squares* (MLS, [80]; also implemented in the PCL) method. However, the source code contained a bug which practically left the surface intact. In addition to fixing this bug, a few other minor bugs were corrected.

A demonstration of an assembly task begins by recognizing the objects on the table. The Kinect sensor is first used to acquire an RGB-D image of the table where the objects reside. The tabletop is then filtered from the image, such that we obtain



(a) ***(b)***

Figure 3.7: ***(a)*** *A partial view of of a 3D model, and* ***(b)*** *its corresponding point cloud. The OUR-CVFH descriptors are computed from the point cloud.*

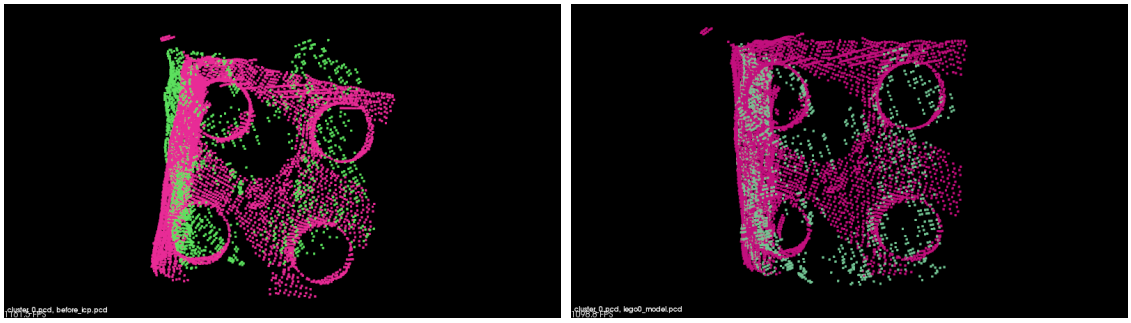
distinct point cloud clusters which correspond to the objects on the table. These segmented clusters are processed one by one: First the recognition software computes OUR-CVFH descriptors for a cluster, and then it finds the closest match from the database containing the descriptors of the model's partial views. Accordingly, the more partial views are used to teach the recognizer, the better are the chances of correctly recognizing the observed point cloud.

After a matching descriptor has been found, the software computes the pose of the descriptor with relation to the Kinect sensor $\boldsymbol{T}_{Kinect}^{Descriptor}$. During the teaching phase of the recognizer we also acquire a transformation matrix from each descriptor to the coordinate system of the object's model $\boldsymbol{T}_{Descriptor}^{Object}$. Therefore, the software exploits the knowledge of the descriptor's pose with respect to the Kinect and with respect to the model's coordinate system to compute the pose of the object with respect to the Kinect sensor $\boldsymbol{T}_{Kinect}^{Object}$ as

$$\boldsymbol{T}_{Kinect}^{Object} = \boldsymbol{T}_{Kinect}^{Descriptor} \boldsymbol{T}_{Descriptor}^{Object}. \tag{3.1}$$

This provides a crude estimate of the pose of an object. However, usually this is not sufficiently accurate for our purposes. Therefore, we further employ PCL's ICP algorithm to gain a more accurate estimation. The ICP algorithm minimizes the difference between two clouds of points by keeping one of the clouds fixed, and translating and rotating the other cloud. In this case, we keep the observed cluster fixed, and attempt to fit the transformed model of the Lego to the cluster as Figure 3.8 displays.

Assembly Operation and Saving it into the Database. As we have defined, an assembly operation consists of an unattached block's movement to the vicinity of another block of Legos, such that they are in contact. In contact means that they blocks are either connected, or touch each other. If at least one pair of nodes share a link in the blocks' graph, the blocks are connected.



(a) ***(b)***

Figure 3.8: *Estimate of a cluster's (green pixels) pose* ***(a)*** *before ICP, and* ***(b)*** *after ICP. The model, which is fitted on the found cluster, is depicted in purple pixels. Clearly the pose estimation is significantly more accurate after ICP.*

After the individual Lego bricks have been initially recognized, the teacher is asked which brick he would like to move. The tracker monitors the movement of the Lego, as the teacher places the brick into its new position. During the tracking the Lego may be moved nearly anywhere. The only restriction is that the Lego be visible at all times — although some occlusion is allowed. Hence, the tracked Lego cannot be put, for example, behind another Lego (as viewed from the Kinect). Moreover, excessive or hasty rotation of a brick may result in failed tracking — the limitations of the tracker are further discussed in Section 5. When the teacher has positioned the Lego into a desired location, he will tell the program to stop tracking the brick. The program will then update the position of the Lego that was moved. However, the tracking system is quite imprecise. Hence, after the tracking has ceased, we must again employ ICP to estimate the pose of the relocated brick accurately.

When the position of a relocated block of Legos is updated, so are the positions of its nodes. The locations of the nodes are then compared to the known locations of the other graphs' nodes by computing their Euclidean distances. If the distance between two nodes is smaller than 12 millimetres, they are considered as joined, and a link is formed between the nodes.[11] Additionally, if Legos are put side-by-side on the tabletop, the distance between any pair of nodes must be smaller than 40 millimetres for the graphs to be considered as united.[12] Note that in a side-by-side configuration there are no links between the nodes in different blocks. Hence, the graph of the resulting structure is not connected. In other words, it is not possible to reach any given node from any other node in the graph by traversing their successive neighbors' links.

We use the 3D models of the bricks in the tracking process. When a brick is moved to a novel location, and the positions of its nodes are updated, so is the model of the newly formed block of Legos. Thus, the model of this larger structure is comprised of the models of the bricks which it contains. Then, later on in the assembly operation, we use this updated model in the tracking of the larger structure.

Each assembly operation is encoded into the database of assembly operations. As Section 3.2.2 describes, each operation is encoded using the adjacency matrices of the Lego blocks before and after merging them, and the transformation matrix between them. The assembly operation step is repeated as many times as the teacher wants, or until all recognized Legos are connected.

To summarize the learning of a demonstrated assembly task: As a precondition, the recognizer must be taught which objects it should identify. The recognizer is then used to identify and estimate the poses of all available Legos, which are presented in front of a Kinect sensor. Subsequently, the teacher tells our program which Lego he wants to relocate. The tracker is used to monitor the movement of the Lego. When the teacher tells the program to quit tracking, the location of the manipulated Lego

[11] Ideally, the distance would be 0 millimetres, as the relocated Lego's undermost nodes coincide with another Lego's uppermost nodes when they are placed on top of each other.

[12] The distance between two nodes in a Lego is 32 millimetres. Thus, ideally, the distance between two outermost nodes in a side-by-side situation should be the same.

is updated, as well as all the other graph structures. If the manipulated Lego is relocated such that is in contact with another Lego, the assembly operation will be saved into a database. This database is then exploited when the observed assembly task is reproduced.

3.3 Reproducing Symbolic Assembly Tasks

After the robot has observed an assembly task and saved all the assembly operations into its database, it can replicate the task. When the robot attempts to replicate the task, it systematically goes through the learned operations in its database. There are two possible types of replications of a task depending on what kind of Legos are supplied to the robot. In an exact replication, the robot is given the exactly same set of Lego bricks to reproduce the assembly task. In this case the robot can replicate the task exactly, as it can perform all the necessary operations using the same bricks as the human teacher did. In an inexact replication the Legos provided to the robot are dissimilar, and thus all the operations necessary to complete the observed task are not found in the database. Thus, when the robot attempts to replicate the assembly operations, it will sooner or later encounter an assembly operation which involves blocks of Legos that it does not possess. The robot, however, does know what kind of an assembly product *should* be produced after it has performed this operation. Hence, in this situation, we use inexact graph matching techniques to find a configuration of currently available Legos that is as similar as possible to the expected assembly product. In the experiments we will use two different kinds of similarity measures to find this similar configuration.

Algorithm 3.1 describes how an assembly task is reproduced. Our program is first supplied with a database that contains all the learned assembly operations. The program will go through the operations one by one. It begins by reading the adjacency matrices of the required initial blocks of Legos, the adjacency matrix of the post-operation product, and the transformation matrix that realises the end product. The adjacency matrices of the initial blocks are then used to determine whether we must execute an exact replication of the assembly operation, or an inexact replication. Since the robot knows what kinds of blocks are available, it goes through them all and compares them to the required initial blocks using one of the two dissimilarity measures. The dissimilarity measure defines a *cost*[13] between each available block of Legos and the required initial blocks. If the required initial blocks are found within the available Legos, the costs for both initial blocks are zero. In this situation the assembly operation will be replicated using the known transformation matrix to place the blocks into the configuration which forms the desired end product. However, if one or neither of the required initial blocks are found, the sum of the costs is higher than zero, and our program executes an inexact replication of the operation.

[13]In the experiments, we will usually refer to the dissimilarity between two graphs as a cost.

Algorithm 3.1 Replicating an assembly task

```
1: procedure REPLICATE(database)
2:     while Successfully read an operation from database do
3:         block_1 ← first required initial block
4:         block_2 ← second required initial block
5:         desired ← desired end product from database
6:         trans ← transformation matrix between initial blocks
7:
8:         while a_i ∈ available Legos do
9:             cost_{1,i} ← dissimilarity(block_1, a_i)
10:            cost_{2,i} ← dissimilarity(block_2, a_i)
11:
12:        L, K ← arg min_{l,k} [cost_{1,l} + cost_{2,k}],   l ≠ k
13:        cost_total ← cost_{1,L} + cost_{2,K}
14:        if cost_total = 0 then
15:            T ← trans                          ▷ This is exact replication
16:        else
17:            T ← inexactReplication(a_L, a_K, desired)
18:
19:        Place a_K on top of a_L according to the transformation matrix T
```

Performing an Inexact Replication. In an inexact replication we choose the initial Legos by minimizing the sum of the costs. As the robot does not know how to place these Legos, we exploit inexact graph matching to find a suitable configuration. The suitable configuration is found by comparing the adjacency matrix of the desired end product to all plausible compositions of the chosen initial blocks. In a manner of speaking, when the robot confronts an inexact replication, it must "improvise" and simulate various compositions to find the best configuration. The best configuration, naturally, is the one which minimizes the dissimilarity (the cost) when compared to the desired end product. The robot will go through all different compositions systematically, in a way in which a small child perhaps would perform the same task: By attempting to place one block on top of the another in each different position. However, there is one significant restriction involved. The robot cannot place two blocks of Legos side-by-side on the tabletop, such that the blocks are disconnected, using inexact replication. This restriction is further discussed in Section 5.

Algorithm 3.2 depicts how the inexact replication of an assembly task proceeds. We have already found the Legos that resemble most the initial blocks, as described in Algorithm 3.1.[14] In the learned assembly operation the second block is always placed

[14]Note that even if one of the initial blocks is available, the program might choose to discard it and use some other block instead. For example, say there are blocks A, B, and C available, and the assembly operation needs blocks α and δ. Now, $cost(\alpha, A) = 0$, and the other found costs are $cost(\alpha, B) = 0.1$, $cost(\alpha, C) = 0.2$, and $cost(\delta, C) = cost(\delta, B) = 0.5$, $cost(\delta, A) = 0.1$. Thus, the cost is minimized when the chosen blocks are $\alpha = B$ and $\delta = A$.

Algorithm 3.2 Inexact replication of an operation

```
1:  procedure INEXACT_REPLICATION(block1, block2, desired)
2:      s ← compute_symmetry(block2)                          ▷ s ∈ {1, 2, 4}
3:      i ← 0
4:
5:      while i < s do
6:          Φ ← the set of configurations when block2 is slid on top of block1
7:
8:          while p_k ∈ Φ do                                  ▷ p_k is a plausible configuration
9:              if p_k is valid then
10:                 cost_{i,k} ← dissimilarity(p_k, desired)
11:                 trans_{i,k} ← getTransformationMatrix(p_k)
12:
13:         block2 ← rotate(block2, 90)          ▷ Rotate block2 90 degrees along z-axis
14:         i = i + 1
15:
16:     I, K ← arg min_{i,k} cost_{i,k}
17:
18:     return trans_{I,K}
```

on top of (or besides) the first block. In the inexact replication we will obey this same order. Hence, in Algorithm 3.2 we will go through only those configurations, where the second block is placed on top of the first block.

Before we begin to simulate all the possible compositions, we determine whether the second block is fully symmetric, semi-symmetric or neither. For example, a 2x2 Lego brick is fully symmetric, and a 2x4 Lego brick is semi-symmetric. If the second block is fully symmetric, we need to do only one pass of placing the second block on top of the first. If the second block is semi-symmetric we need two passes, and not symmetric at all, four passes. Hence, symmetry reduces the computation time slightly. After each pass, the second block is rotated 90 degrees along its z-axis (which points upwards). One pass is like a convolution operation: We slide the second brick on top of the first brick, such that each possible configuration is simulated. Figure 3.9 demonstrates how the program goes through the configurations. When the program finds a valid configuration, it computes the dissimilarity between the found configuration and the desired end product. A valid configuration means a composition where each node is connected to a maximum of one node in another brick. After all possible configurations have been simulated, and their costs computed, the program chooses the configuration with the minimum cost. According to the used similarity measure, this is the configuration that resembles most the desired end product.

In order to compute the cost of a plausible configuration, we need its adjacency matrix. To get the adjacency matrix, we need to simulate the composition (as Figure 3.9 displays). In the simulation the program places the models of the blocks into a

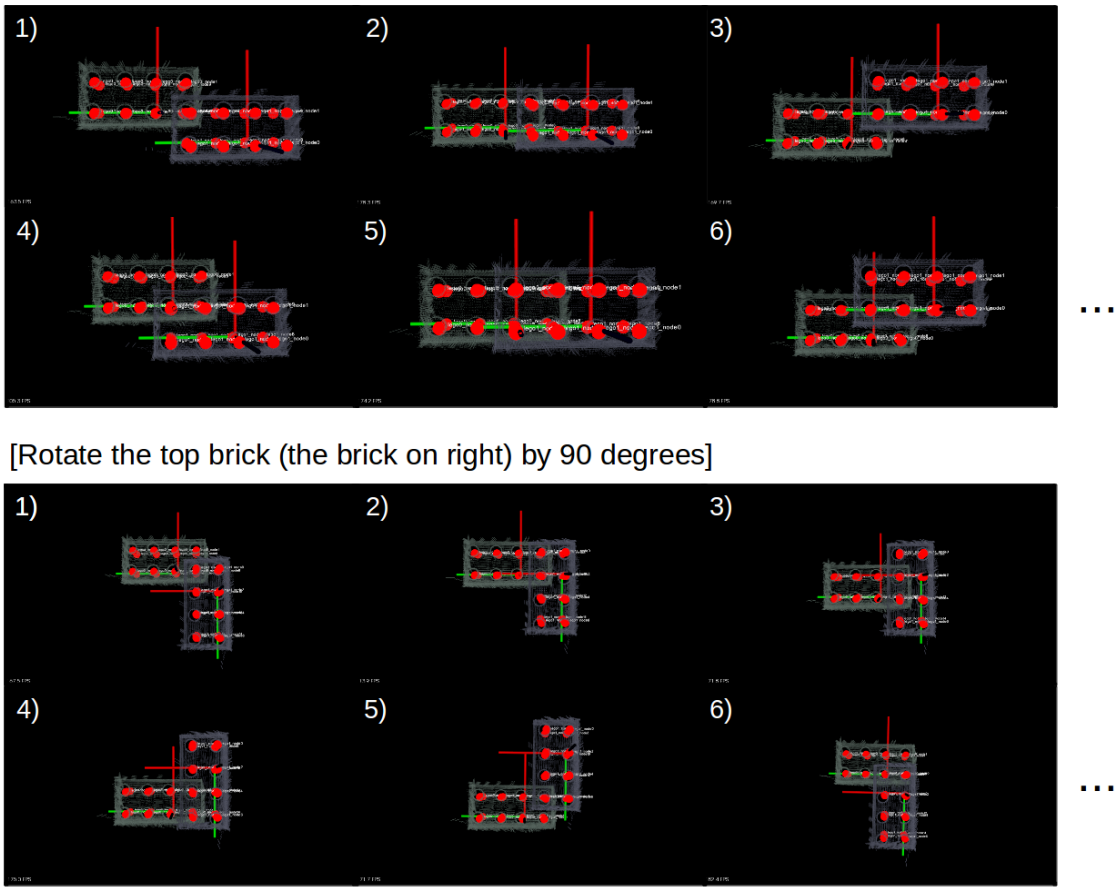


Figure 3.9: *This figure depicts how the program browses through all possible configurations. The upper image depicts the first six configurations. After the program has processed all configurations when the bricks are directed as shown, the top brick is rotated by 90 degrees along its z-axis. Then the program again starts the convolution-like sliding of the top brick: Again the first six configurations are shown. Note that here the top brick is semi-symmetric, thus only 2 passes of the sliding operation are required to cover all plausible configurations.*

plausible configuration, and computes its adjacency matrix. Therefore, in order to get the simulated composition, we need the transformation matrix which realises the configuration. Hence, when we compute a cost for a plausible configuration, we have to also compute the transformation matrix. After the program has browsed through all possible configurations, it simply chooses the configuration — and transformation matrix — associated with the lowest cost. This transformation matrix tells us how to place the second block of Legos on top of the first block of Legos such that the configuration with the minimum cost is achieved.

Note that once one inexactly replicated operation is performed, the effects cascade in the assembly task, and all the subsequent operations which involve this product must also be inexactly replicated.

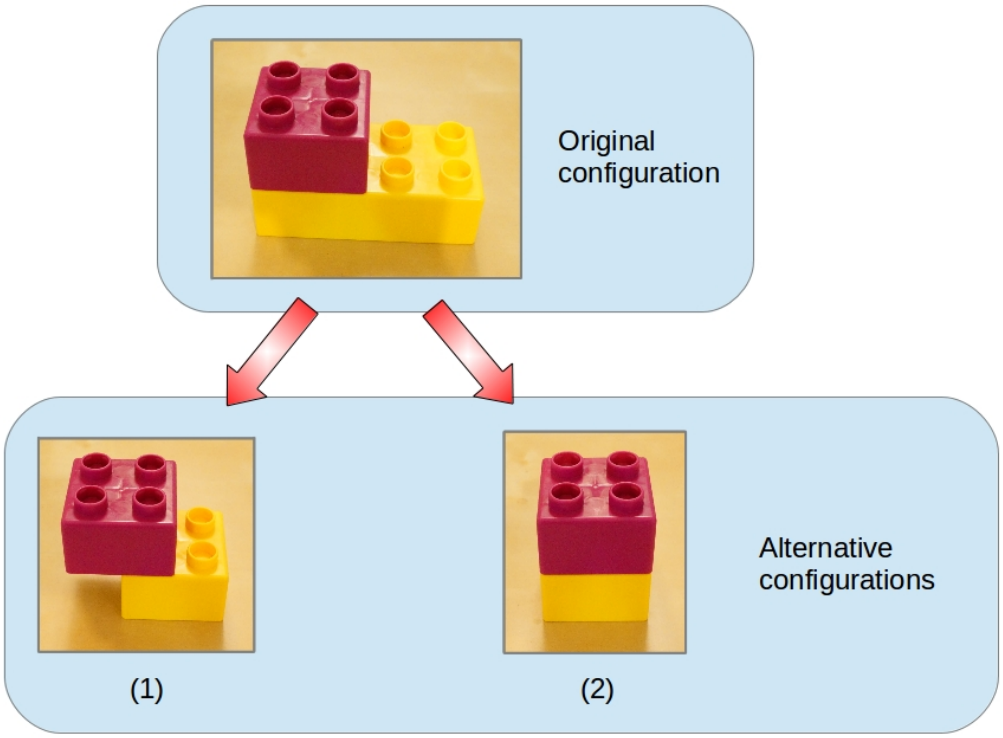


Figure 3.10: *It is ambiguous, even for a human, which alternative configuration resembles most the original configuration on the top.*

The objective of an inexact replication is to generate an alternative configuration of the available Legos, that resembles as much as possible the configuration of the learned assembly operation. Ideally, we would like to have a similarity measure that seems intuitive at some level — for example, a measure that matches the bottom left configuration to the original configuration in Figure 3.10. However, we do acknowledge that this is a troublesome issue, because it is ambiguous when two Lego structures are similar. Take for example Figure 3.10. Here the configuration of Legos that resembles most the construction on top — the configuration with one 2x2 brick and one 2x4 brick — is problematic to obtain. Below the original configuration there are two alternative constructions with two 2x2 bricks. Either one of these alternatives could be seen as the most similar configuration to the original construction. In the alternative **(1)** the overall shape of the structure resembles more the original construction, and the ratio of occupied nodes to unoccupied nodes is the same. This allows for another Lego to be joined besides the top Lego, as is the case in the original construction as well. The alternative **(2)**, on the other hand, has the same height as the original construction, and the overall stability in this structure is firmer, as in the original construction. However, the shape does not allow the placement of another Lego besides the top Lego. Note that these are not the only plausible configurations. In the same manner we could claim that a configuration where the two 2x2 Lego bricks are side by side resembles most the original construction.

To summarize the reproduction of assembly tasks: Each assembly task is composed of a sequence of assembly operations, which are saved into a database. Each operation is encoded into the database as the adjacency matrices of the initial blocks, which are required to perform the assembly operation exactly, the adjacency matrix of the assembly product, and the transformation matrix of the initial blocks that realise the assembly product. When the robot replicates an assembly task, it goes through the assembly operations one by one. In the beginning of each assembly operation, the robot browses through the available Legos to see whether it can simply replicate the learned operation. If the Legos which are required to execute the assembly operation are available, the robot uses the saved transformation matrix to arrange the available Legos in a configuration that forms the desired end product. However, if the required initial blocks are not available, the robot chooses assembly parts which are most similar to them. Then the robot exploits the desired assembly product's adjacency matrix, and finds a configuration of the chosen Legos that resemble most this desired adjacency matrix. The similarities are evaluated using a dissimilarity measure. In the experiments we use the Laplacian dissimilarity measure and the JoEig dissimilarity measure, as presented in Section 2.3.1.

There are two critical restrictions that influence greatly the inexact replication tasks. Firstly, the robot replicates the same number of operations as it observes. This means that the robot always uses the same number of Legos as the human teacher does in the observed task. However, in some occasions it would be beneficial to use less or more Legos to build a maximally similar structure. Secondly, as mentioned above, when the required initial blocks are not available, the robot chooses greedily the assembly parts which are most similar to them. Instead of choosing the assembly parts greedily, the robot could also go through all available Legos and their configurations to find the structure that is most similar to the desired structure. The effects of these restrictions are discussed in Section 5.

4 Experiments and Results

We present three different sets of experiments, that examine the viability of our method. In general, each single experiment consists of two parts: a teaching phase and a replication phase. In the teaching phase, a human teacher demonstrates the assembly tasks in front of a Kinect sensor. In the replication phase, the robot attempts to replicate the observed assembly task. These replications are not executed using a real robot, but instead they are run as simulations. When we say that a robot replicates an observed assembly task, we mean that our system generates an assembly plan, or basically a simulation, that accomplishes the learned task. These simulations are visually assessed to determine the quality of the assembly plan.

The experiments aim to answer two questions: 1) Does our proposed method learn to replicate the observed assembly tasks successfully, and 2) how does our method cope with uncertain knowledge of the assembly operations? The first set of experiments, the *exact replication experiments*, aim to answer the first question. Then, the last two sets of experiments, the *inexact* and the *greedy construction experiments*, aim to answer the second question. The uncertainty is imposed on the experiments by either replacing some of the Legos the teacher used in the demonstration, or by preventing the robot from observing the assembly operations. Furthermore, the inexact and greedy construction experiments examine the behaviour of the dissimilarity measures presented in Section 2.3.1. A desirable dissimilarity measure would produce a structure that is similar to the learned one, even with incomplete knowledge of the assembly parts or assembly operations.

Essentially, in all experiments the robot attempts to replicate what a human has demonstrated. In the exact replication experiments, the robot is supplied with the same set of Legos as the teacher had in the demonstrations. Thus, the robot will replicate precisely the same assembly operations as the teacher demonstrated. In the inexact replication experiments, the robot is provided with a different set of Legos. In other words, the robot has a different number of 2x2 and 2x4 bricks available than the teacher had in his demonstration. In effect, we randomly choose to replace one or more 2x2 bricks with a 2x4 brick, and vice versa. In these experiments, the robot cannot exactly replicate the learned assembly task. Instead, its objective is to produce a structure, that is as similar as possible to the observed end product of a given assembly operation. Lastly, in the greedy construction experiments, the robot is supplied only with a graph of the final assembly product. Again, the robot's task is to use the given set of Legos to build a construction that resembles the observed one. This set of Legos is the same set as the teacher had in his demonstrations. The robot replicates the assembly operations inexactly in these experiments also, since it does not have information of them. During each stage in the assembly, the robot joins two assembly parts such that the resulting structure is as similar as possible to the final assembly product. Thus, the generation of the assembly plan is based on a greedy "choose-the-best-next-step" search.

The dissimilarity measure is used twice during each assembly operation in the inexact

replication experiments. First to find the initial blocks required to execute an assembly operation, and then to produce a configuration of these initial blocks that resembles the desired end product. However, in the greedy construction experiments the dissimilarity measure is used only once during each assembly operation: To find a configuration of the available blocks that is as close as possible to the final graph of the assembly product. In addition to computing the dissimilarity measures, we also judge the similarity of assembly products by visual scrutiny — two structures are similar if they have a similar shape.

A few critical assumptions apply to the experiments. Firstly, the teacher is always expected to succeed in his demonstrations. Likewise, the robot is always assumed to successfully replicate the operations in the simulations. In other words, both the teacher and the robot always manages to move a Lego block to a desired location, without breaking the existing construction.

Secondly, the recognition and tracking processes restrict the way we place and move Legos on the table. In the recognition step a Lego brick cannot be greatly occluded, and thus the Legos should be situated an adequate distance apart from each other. In the tracking step the Lego block has to be always visible to the Kinect sensor, although a reasonable amount of occlusion is allowed. This also limits the composition of the Lego aggregate, since a small block of Legos cannot be placed behind a larger block of Legos, otherwise the tracker will lose sight of the target.

Thirdly, in addition to the locations of the Legos, also the number of the Lego bricks on the table is limited. In the recognition step the tabletop is filtered, and the remaining clusters are segmented and recognized as Legos. If there are too many Legos on the table, the algorithm cannot find the tabletop. Consequently, the filtering of the tabletop fails, and the remaining clusters do not represent the bricks. Furthermore, the Legos should be in an upright position, otherwise they will not be recognized. Figure 4.1 displays a beginning of a learning phase. All the Lego bricks are clearly visible to the Kinect, far enough from each other, and in an upright position.

4.1 Exact Replications

In these experiments the robot has at its disposal exactly the same set of Legos as the teacher had during the demonstrations. Therefore, the robot can easily replicate the same assembly operations which it observed. We present three elementary experiments which highlight some of the properties of our proposed approach.

In each experiment we present the learning phase and the replication phase. These are color coded such that images of the observed demonstration are on a yellow background, while images of the replications are on a reddish background, see for example Figure 4.2. In the learning phase we show the whole set-up — that is, the Kinect and the Legos — at each stage in the assembly task. In the replication phase we show images of the assembly product after each stage as produced by our method.

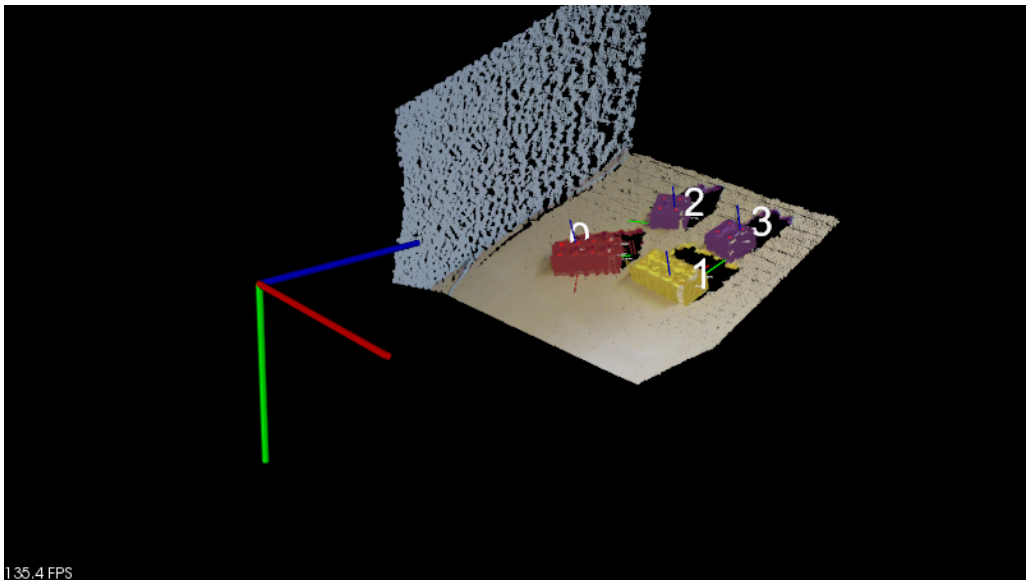


Figure 4.1: *The beginning of a learning phase: The program is asking which Lego brick the user intends to move. The Kinect overlooks the table where the Lego bricks are situated. The large axes in the front are the axes of the Kinect's coordinate frame. The distance from the Kinect to the bricks was around 50 – 100 centimeters in the experiments.*

In the first experiment we also show images of how the Kinect sees the setting in the learning phase, and how our system generates the simulated assembly in the replication phase.

First Exact Experiment

This experiment (Figure 4.2) demonstrates a straightforward assembly, where the Lego bricks are placed on top of each other. Each brick is sequentially placed on top of the growing aggregate. This is the simplest way to create an assembly product.

In the learning phase, the top row illustrates the experimental set-up. The bottom row of the learning phase shows the scene as viewed from the Kinect. These images also show how the Legos are represented as graph structures, and display the bricks' models which are used for tracking.

Our program outputs the images with the black background in the replication phase. These images illustrate how and where one should put the available Lego bricks to replicate the learned assembly task. The replication with real Legos is conducted by a human following these instructions. In the following exact replication experiments we show only these human-made replications. These images with real Legos show the structures of the replicated assembly products better than the simulations with the black background.

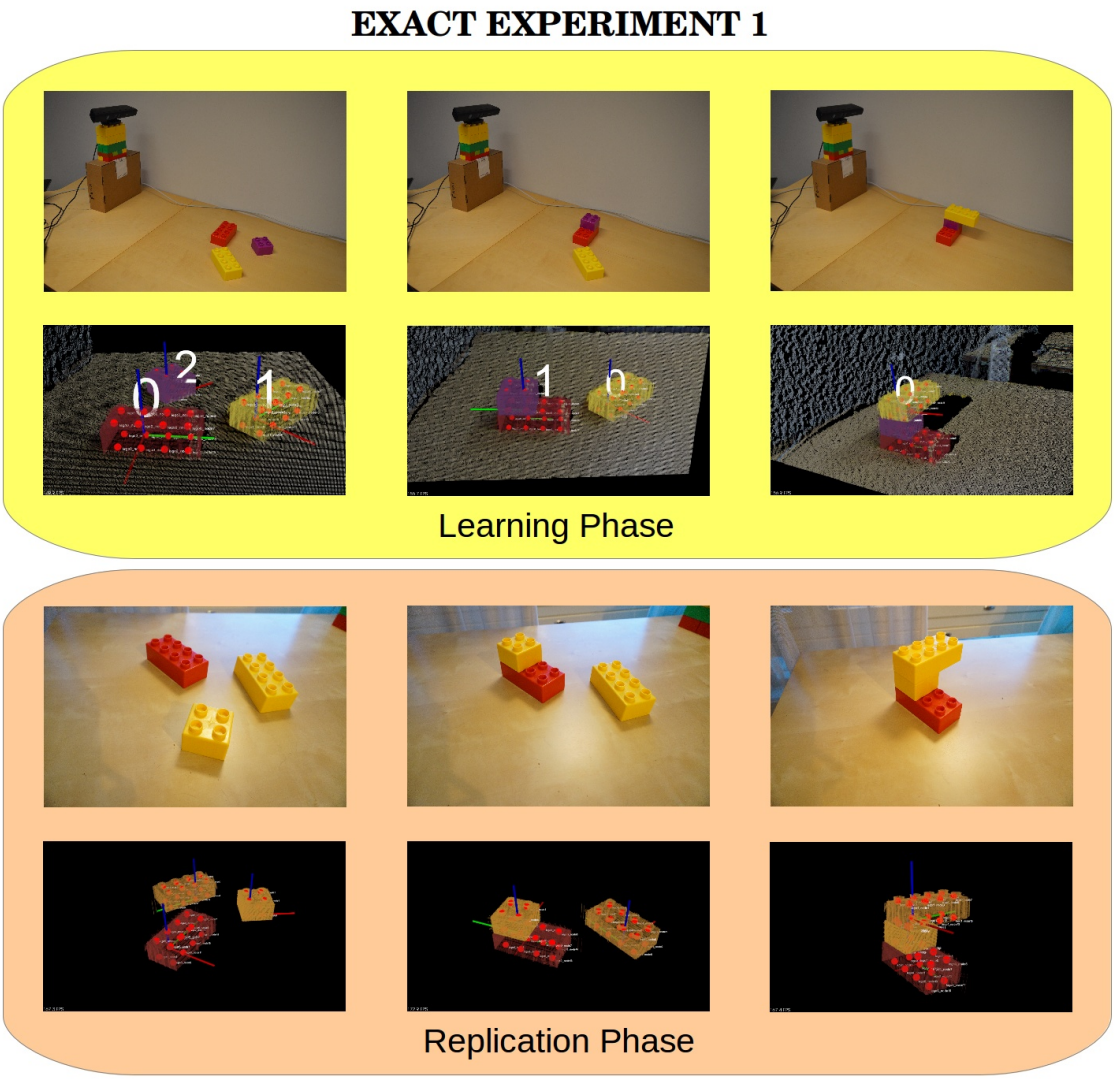


Figure 4.2: *This figure depicts the learning and replication phases. Each column represents a stage in the assembly task, such that the rightmost images show the finished assembly product. In this experiment, the Lego bricks are simply joined sequentially into one structure.*

Second Exact Experiment

In the exact replications, the blocks of Legos can also be placed side-by-side on the tabletop. Figure 4.3 shows how a small 2x2 brick and a large 2x4 brick are first placed side-by-side, and then they are connected using another 2x2 brick. After the first assembly operation, the side-by-side arranged Legos form a single graph, but the nodes of the separate bricks do not share any connections. Only after the third brick is placed on top of the existing structure, the graph becomes connected.

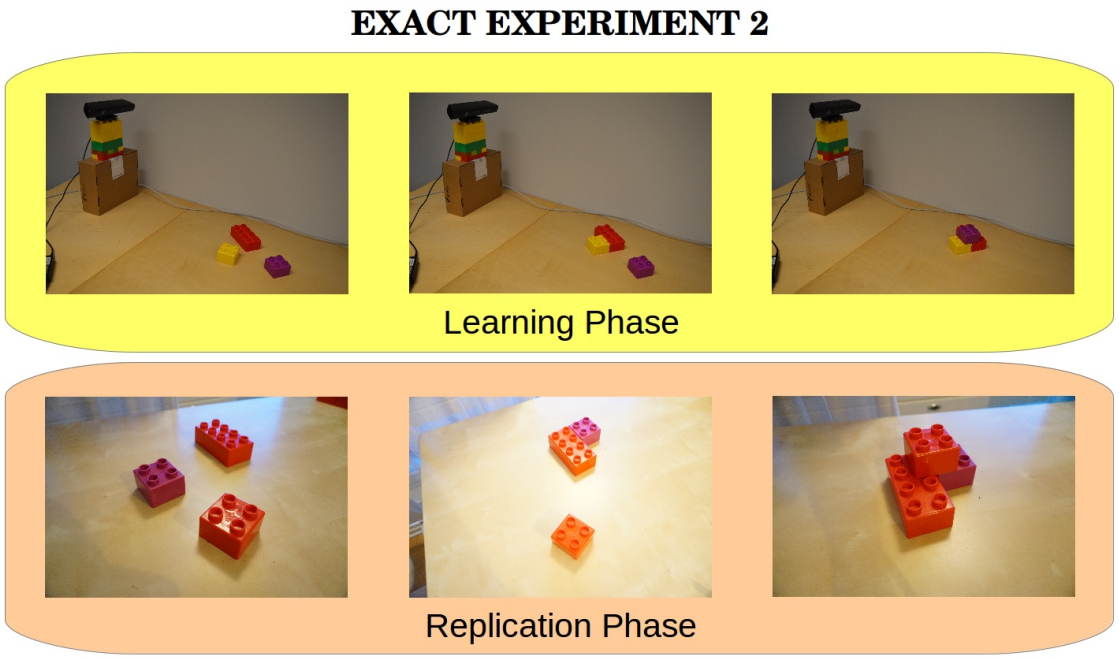


Figure 4.3: *Both the learning phase and replication phase are shown for the second inexact experiment. The assembly progresses from left to right. The teacher can move blocks of Legos side-by-side on the table, and our method can reproduce the assembly product.*

Third Exact Experiment

In this experiment we first produce two subassemblies consisting of two Lego bricks, and then join these two subassemblies to obtain the final assembly product (Figure 4.4). This experiment demonstrates that the assemblies can be built in any order. The human teacher does not need to construct one structure by sequentially joining single bricks into it, but instead he can build two or more structures first, and then join all these subassemblies together.

These three simple exact experiments exhibit the typical characteristics of assembly tasks. Clearly our method can replicate these tasks. Further, our approach can replicate any task the teacher demonstrates, as long as the robot has the exactly same set of Legos available. Next we delve into a more difficult scenario, where the robot is supplied with a different set of Legos than the one it observed in the demonstrations.

4.2 Inexact Replications

Unlike in the exact experiments, this time the robot is provided with an incomplete set of Legos. More specifically, the incomplete set of Legos differs from the complete set of Legos in that some of the bricks are replaced. Either one or more 2x2 bricks

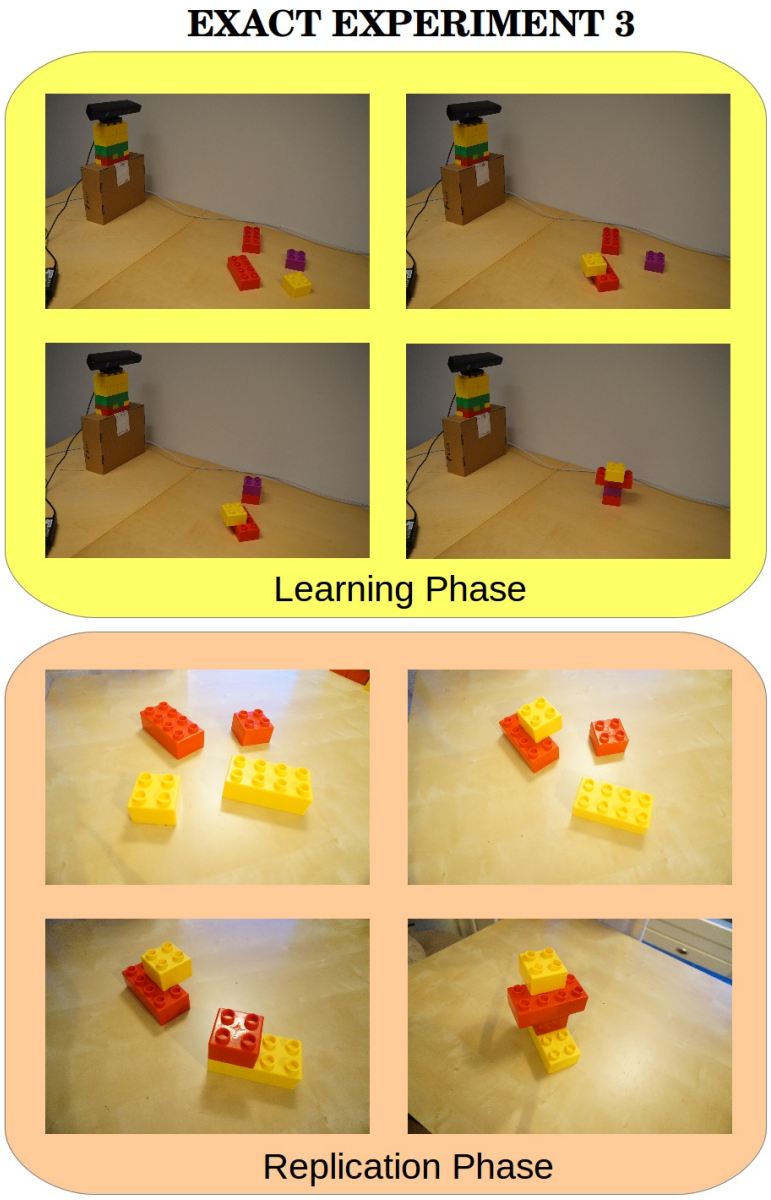


Figure 4.4: *The learning and replication phases for the third exact experiment. The assembly progresses from left to right, top to bottom. The Lego bricks may first be merged into smaller subassemblies, which are then joined to form the final assembly product.*

are replaced with 2x4 bricks, or conversely, one or more 2x4 bricks are replaced with 2x2 bricks. However, the number of Legos remains the same, and hence the robot will perform the same number of assembly operations as it observed.

Essentially, there are two alternative ways in which the inexact replication may begin. Either a smaller 2x2 Lego brick is substituted into a larger 2x4 brick, or vice versa. The former situation is easier to handle, since a smaller graph is replaced with a larger one, thus increasing the number of nodes — the places where another Lego

brick can be connected. The opposite situation, where a larger graph is substituted into a smaller one, is more difficult since we lose nodes.

The aim of these inexact — and greedy construction — experiments is to build similar structures, even though the provided set of Legos is incomplete. We attempt to build similar structures by using two different dissimilarity measures, as explained in Section 3.3. The first measure is the Laplacian dissimilarity measure, which uses the eigenvalues of the graphs' Laplacian matrices to compute the difference between the graphs. The second measure is the JoEig dissimilarity measure, which uses the eigenvalues and eigenvectors of the graphs' adjacency matrices to compute the discrepancy. The inexact and greedy construction experiments examine the behaviour of these dissimilarity measures. Specifically, we want to find out whether these measures produce different structures, and also do the structures resemble the originally observed ones.

First, we introduce three simple and straightforward experiments. These experiments contain only three Lego bricks, hence each experiment is replicated with all possible sets of Legos — except the set of Legos that was used in teaching. Moreover, each set of Legos is replicated with both dissimilarity measures in order to see how they differ from one another. Generally, in the following figures of the experiments, the number of 2x4 Lego bricks increases from left to right (see the explanation in the caption of Figure 4.5).

Afterwards, we present four complicated experiments, which contain a large number of Legos (six to nine) in diverse compositions. For each experiment we show the original assembly product as demonstrated by the teacher, and five inexact replications using different sets of Legos. We present either an image of the assembly product as performed by a human according to the robot's orders, or an image of the simulation, depending on whether the assembly is stable enough to be constructed or not.[15] Moreover, in an attempt to show the configurations of the Legos unambiguously, the images may not be from the point of view of the Kinect, but from a different angle. Also, as in the first three inexact experiments, the number of 2x4 bricks increases from left to right in the figures of the results.

We also display the dissimilarity measure for each inexact replication. As was mentioned earlier, we will generally call the dissimilarity measure a cost. Furthermore, we use the following three terms interchangeably, as they mean the same thing:

$$\text{Maximize similarity} = \text{Minimize dissimilarity} = \text{Minimize cost.}$$

First Inexact Experiment

The learned construction consists of three 2x2 bricks in a simple tower configuration, see Figure 4.5. In this experiment, we replace each 2x2 brick one by one with a

[15]There are two distinct manifestations of instability: Either the structure cannot stand upright, or then it doesn't hold at all, in any position.

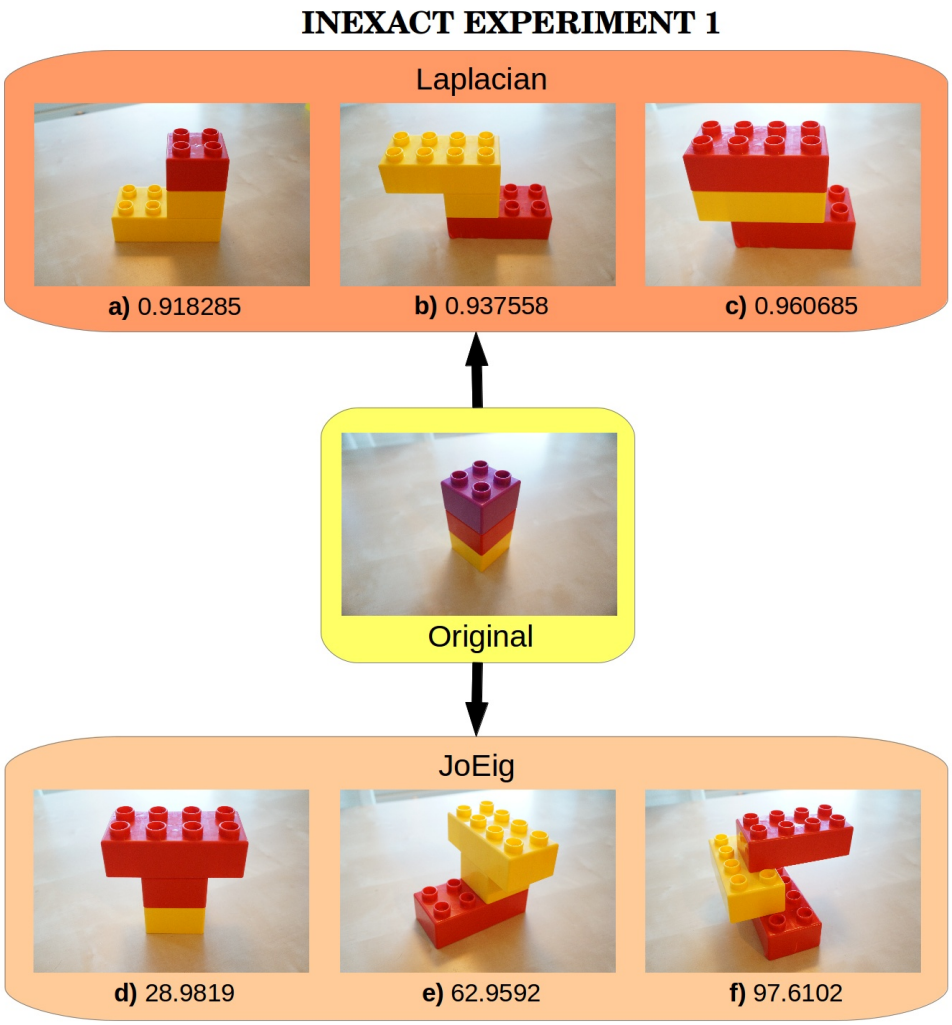


Figure 4.5: *The configurations in* ***a*** *-* ***c*** *represent the inexact replications when we use the Laplacian dissimilarity measure. The configurations in* ***d*** *-* ***f*** *represent the inexact replications with the JoEig dissimilarity measure. The number of replaced 2x2 bricks increases from left to right, such that in* ***a*** *and* ***d*** *we have replaced one 2x2 with a 2x4, in* ***b*** *and* ***e*** *we have replaced two 2x2s with two 2x4s, and finally, in* ***c*** *and* ***f*** *we have replaced all three 2x2 bricks with 2x4 bricks. Each upcoming figure that displays the results of an experiment follows this same convention: The number of 2x4 bricks increases from left to right. The values under each configuration represent the dissimilarity between the replicated structure and the originally learned structure.*

larger 2x4 brick. Then we examine how both dissimilarity measures reconstruct the learned structure with these incomplete sets of Legos. Figure 4.5 presents the final assembly products of the sets with both the Laplacian and the JoEig dissimilarity measures. The measured cost is displayed underneath each final assembly product — the higher the dissimilarity, the further apart the assembled product is from the originally learned structure.

When only one 2x2 brick is changed to a 2x4 (**a** and **d** in Figure 4.5), the last steps are performed differently.[16] In the observed demonstration the teacher first puts two 2x2 bricks on top of each other, and then moves this structure on top of the last single 2x2 brick. With the Laplacian measure the assembly advances similarly: A tower of two 2x2 bricks is put on top of the last 2x4 brick. With the JoEig measure, however, the last 2x4 brick is put on top of the tower of 2x2s. In other words, JoEig finds the cost of replacing the tower of two 2x2 bricks with a 2x4 brick smaller than replacing the missing 2x2 brick with a 2x4 brick. This occurs because we minimize the total cost of choosing the initial blocks for an assembly operation. Therefore, we might choose a "wrong" block as an initial block even though a correct one is available, if the total cost of both initial blocks is minimized this way.

Figure 4.5 shows that when only one 2x2 brick is replaced with a larger 2x4 brick, the overall shape is quite similar to the original structure. But already when two smaller bricks are replaced with 2x4 bricks, the shapes of the inexactly replicated assemblies start to diverge from the original one. Especially when all 2x2 bricks are replaced with 2x4 bricks, the JoEig dissimilarity measure produces a peculiar assembly product: It is difficult to see the resemblance of this structure with the original structure. Additionally, the first assembly operation in this particular inexact replication consisted of placing the red 2x4 brick on top of the yellow 2x4 brick, which results in an unstable structure. The final assembly is stable however. Curiously, the JoEig measure is able to place a 2x4 brick on top of a 2x2 brick rationally (**d** in Figure 4.5), but fails to put a 2x4 brick on top of another 2x4 brick in a reasonable manner (**f** in Figure 4.5).

Unsurprisingly, the dissimilarity increases as we change more 2x2 bricks into 2x4 bricks. In fact, in this experiment both dissimilarity measures increase approximately linearly with the number of replaced Legos.

Second Inexact Experiment

This experiment still poses a relatively simple problem, since we replace the smaller 2x2 bricks with larger 2x4 bricks (see Figure 4.3). Still, it is more difficult than the first experiment, because the structure is in two layers. If one of the bricks on the top is placed on the bottom Lego, such that all its nodes are occupied, then there is no space left to place the last brick. Then the only option is to place the last Lego on top of the second, and the final structure will consist of three layers.

Let us first examine the results of the Laplacian dissimilarity measure in Figure 4.6. If only one 2x2 is switched to a 2x4 (**a**), the replication is close to the original one. However, the next configurations, **b** and **c**, are in three layers and rather far from the original configurations — when judged by visual scrutiny. Note that the configuration in **c** is exactly what we would have wanted in the previous experiment, but neither measure could produce it there.

[16] Note that these steps are not displayed in the figure.

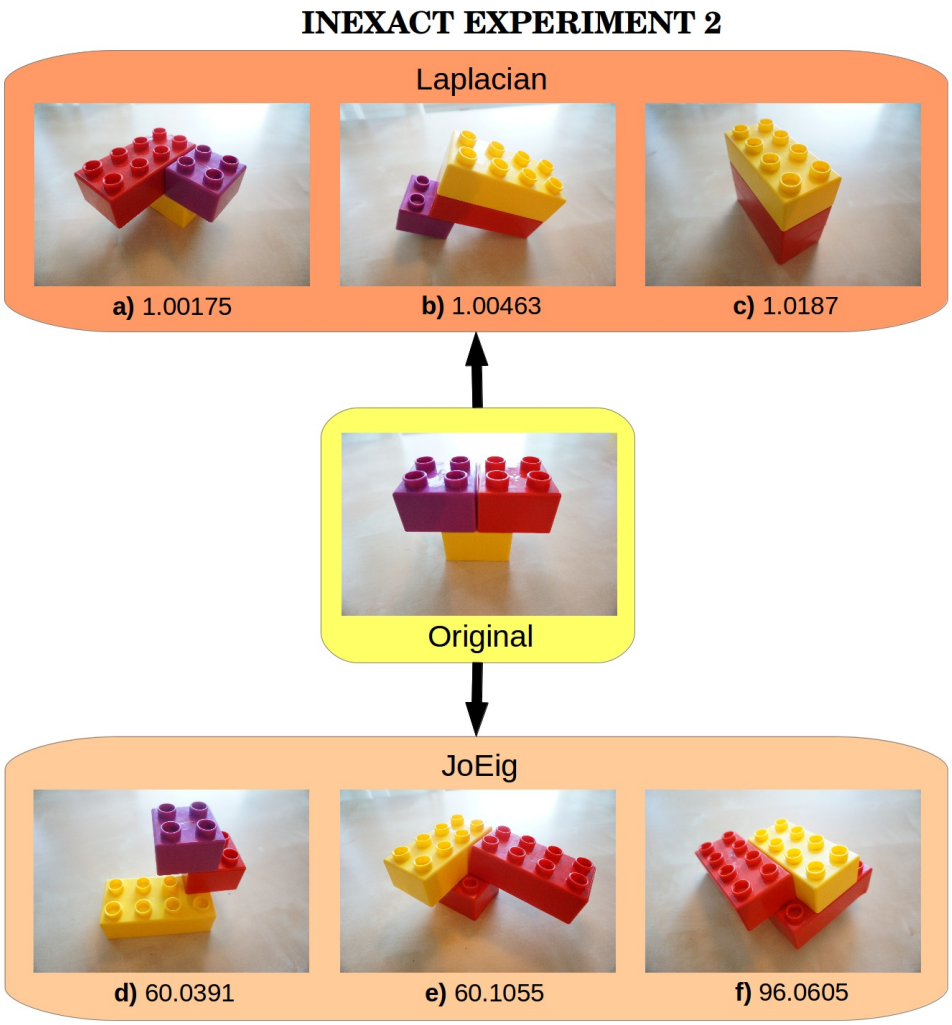


Figure 4.6: *The configuration in experiment 2 poses a more difficult problem than experiment 1, but is still quite simple. The images in* ***a*** *-* ***c*** *are the structures replicated using the Laplacian measure, and images* ***d*** *-* ***f*** *are the structures replicated using the JoEig measure. The values represent the dissimilarity between the replicated and observed structure. The Lego sets used in the replications are formed as described in the caption of Figure 4.5.*

Then, in the JoEig measure, **e** resembles the original configuration more than the others. In replication **d** we see a typical placement of Legos. As this and the upcoming experiments unveil, it is not uncommon for either dissimilarity measure to join two bricks to each other such that they are connected only from the corners. We will discuss this behaviour more in Section 5.

In this experiment, the cost does not increase linearly with the number of replaced bricks. The cost is much higher in the last configurations (**c** and **f** in Figure 4.6) with both measures. As we do the exactly same changes in the first experiment and this experiment (replace three 2x2s with three 2x4s), the adjacency matrices of the final

products are of the same size. Since the cost grows linearly in the first experiment but not this one, we can conclude that the number and placement of the links affect the cost.

Third Inexact Experiment

This is the last experiment of the relatively simple problems. However, now the original configuration includes a 2x4 brick, which can be replaced with a smaller 2x2 brick. Hence, we lose some nodes in this replacement, and it is generally more difficult to obtain a construction that resembles the original structure.

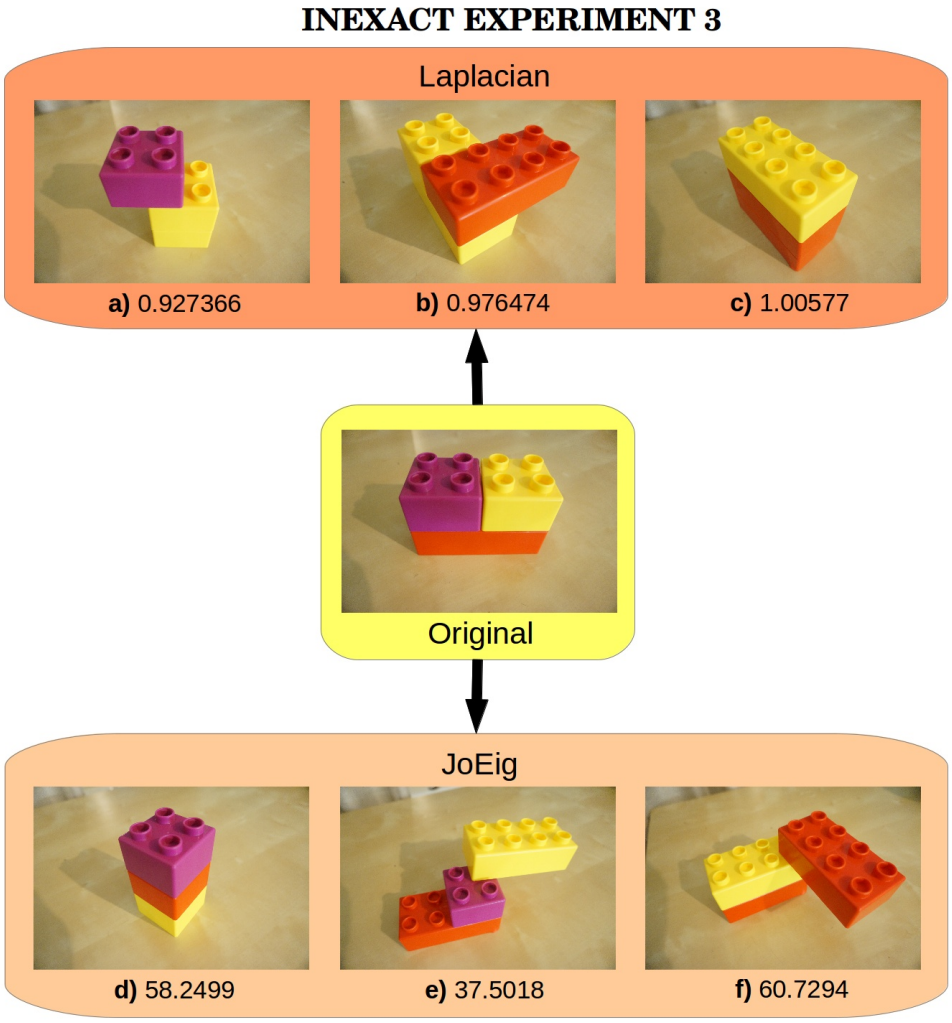


Figure 4.7: *This experiment demonstrates a more difficult case, where a 2x4 brick is replaced with a smaller 2x2 brick (in images* ***a*** *and* ***d****). The images* ***a*** *-* ***c*** *present the structures built using the Laplacian dissimilarity measure, while the images* ***d*** *-* ***f*** *present the structures built using the JoEig dissimilarity measure. The values represent the dissimilarity between the replicated structures and the originally learned structure.*

As Figure 4.7 displays, when the 2x4 brick is replaced with a smaller brick, neither of the dissimilarity measures can manage the situation. However, the configurations in **b** and **c** are similar to the original one. The same applies to the configuration in **f** also, if we disregard the red 2x4 brick on the top.

Fourth Inexact Experiment

The last four experiments deal with more complex structures. In this experiment we use three 2x2 bricks and three 2x4 bricks, and the number of 2x2 bricks increase by one in each of the following experiments. We replicate these experiments with five different sets of Legos, where we have randomly replaced some of the original Legos with a smaller or a larger brick. The results are displayed in the same manner as in the previous experiments.

In this experiment we take advantage of the length of a 2x4 brick, which is placed on top of two other 2x4 bricks, see Figure 4.9. On top of the far end of the red 2x4 brick is a 2x2 brick. Also, the order of assembly operations is crucial in this task: If the Legos are placed in a wrong order, the structure will collapse.

In this experiment, all the structures built using the Laplacian measure are either unstable, or include an impossible assembly operation. In Figure 4.9 **b** the last assembly operation is impossible in practice without disassembling some parts of the built structure first (see Figure 4.8). Nevertheless, the final configuration is valid.

The configurations in Figure 4.9 produced using JoEig, on the other hand, are all stable except for **g**. The lowest costs for both similarity measures occur when one 2x4 brick is replaced with a 2x2 brick (**c** and **h**). However, for both measures the converse situation where a 2x2 brick is changed to a 2x4 brick produces a more intuitively similar structure (configurations **d** and **i**). A rather similar structure is also the configuration in **b**. Here the overall shape is close to the learned assembly product, and, as in the learned structure, the bricks are in four layers.

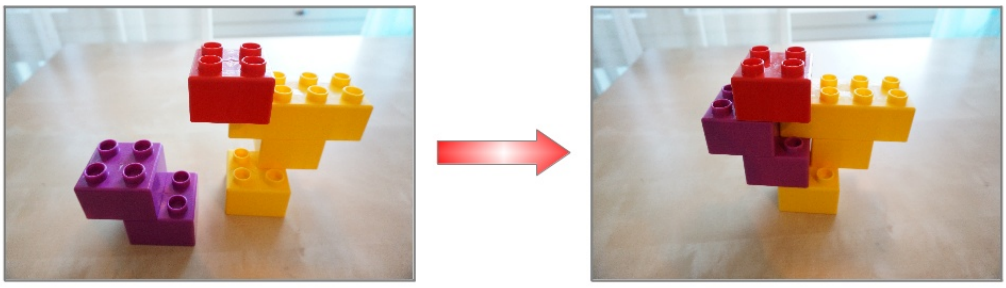


Figure 4.8: *The last assembly operation when replicating* **b** *in Figure 4.9 is impossible in practice, although the final structure is valid. The red 2x2 brick on top of the structure should be removed before the purple block of Legos can be inserted.*

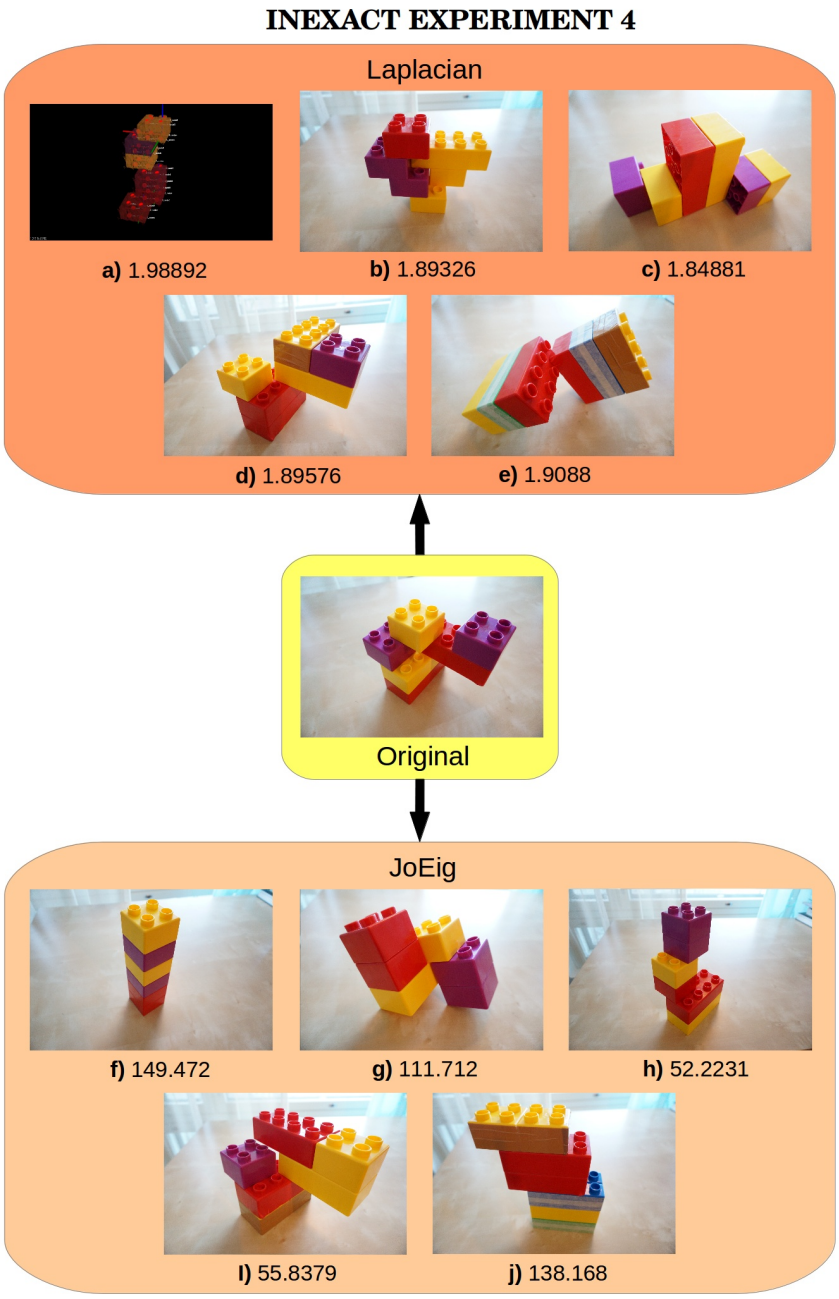


Figure 4.9: *In this experiment we take advantage of the length of the red 2x4 brick, which supports a purple 2x2 brick. The original structure consists of three 2x2 bricks and three 2x4 bricks. The images* ***a*** *-* ***e*** *depict the structures which were replicated using the Laplacian measure, and the images* ***f*** *-* ***j*** *depict the structures which were replicated using the JoEig measure. The set of Legos, which is provided to the robot, is different in each replication, such that the number of 2x4 bricks increases from left to right. In other words, the least number of 2x4 bricks are in the replications* ***a*** *and* ***f****, whereas the most 2x4 bricks are used in the replications* ***e*** *and* ***j****. The values represent the dissimilarity between the replicated structures and the originally learned structure.*

When using only 2x2 bricks to replicate the original structure, both measures seem to favour a tall, tower-like structure. Evidently the system has problems creating wide structures when using only smaller bricks. This is an anticipated consequence, since the inexact replication scheme is not able to place blocks side-by-side on the tabletop. Hence, there remains only one way to build, upwards.

Fifth Inexact Experiment

In this experiment we first move the red and yellow 2x4 bricks side-by-side on the tabletop, and then connect them using a 2x2 brick. Afterwards the whole wide structure is placed on top of another 2x4 brick. This experiment demonstrates the known problem of placing bricks side-by-side: When there are less than two 2x4 bricks available, our method cannot use exact replication, but inexact replication does not know how to place Legos side-by-side on the tabletop. This problem affects the configurations **a** and **f** in Figure 4.10, where all 2x4s are replaced with 2x2s. In these situations the system starts to build a tower-like structure with both dissimilarity measures. The final structures are almost the same, although the structure by JoEig is not stable, and the structure constructed using the Laplacian measure almost collapses.

What is noticeable, is that all structures in Figure 4.10 built using JoEig are unstable — even **g** and **i** collapse at some point, even though they hold for a few seconds. Furthermore, in **j** the bottom 2x4 Legos are not even connected to each other. Three out of the five Laplacian structures, on the other hand, are stable. In **e** there was one stage in the construction which resulted in an unstable configuration, but the end result is nevertheless stable. Structures **d** and **e** are even more stable than the original construction.

For both similarity measures, the situations where one or two 2x2 bricks are replaced with larger ones (**c** – **d** and **h** – **i** in Figure 4.10) are fairly similar to the original one. The configurations in **c** and **h** are the same, but **c** does not have the lowest cost of the Laplacian structures, whereas **h** has the lowest cost of the structures built using JoEig.

At this point, we begin to see that the size of the final structure affects the Laplacian dissimilarity measure. The more Legos there are in the final structure, the higher the cost seems to be.

Sixth Inexact Experiment

This experiment also contains the problem of setting the "base" of two 2x4 bricks side-by-side on the tabletop (although this base is not clearly visible in Figure 4.12). Furthermore, this experiment demonstrates the problem of occlusion. Figure 4.11 presents the last stage of the observed assembly task from the point of view of the Kinect. Here we have two Lego structures, one of which contains a relatively high

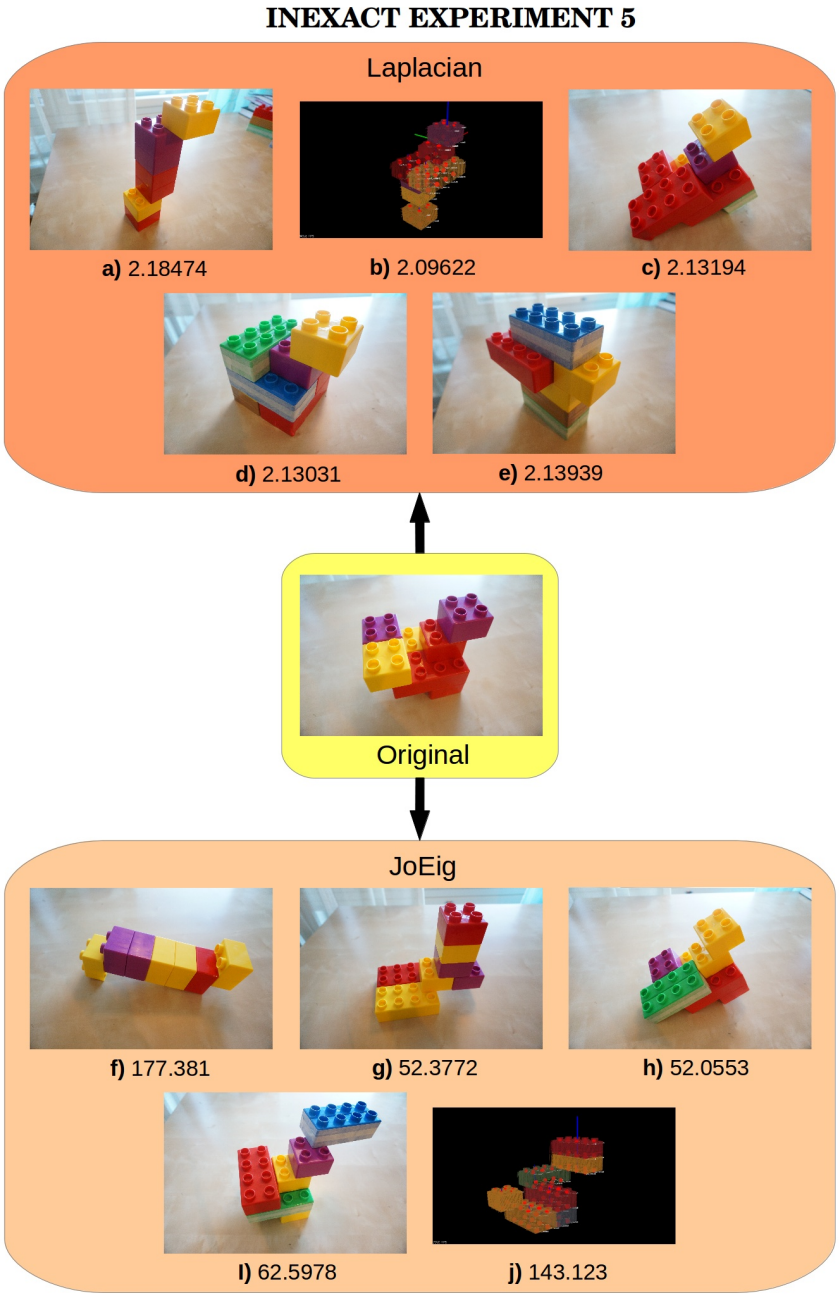


Figure 4.10: *This experiment demonstrates the known problem of setting Legos side-by-side on the tabletop. The original structure consists of three 2x4 bricks and four 2x2 bricks. The structures* ***a*** *-* ***e*** *are replicated using the Laplacian dissimilarity measure, while the structures* ***f*** *-* ***j*** *are replicated using the JoEig dissimilarity measure. The number of 2x2 and 2x4 bricks changes in the replications as is explained in the caption of Figure 4.9. The values represent the dissimilarity between the replicated structures and the learned structure.*

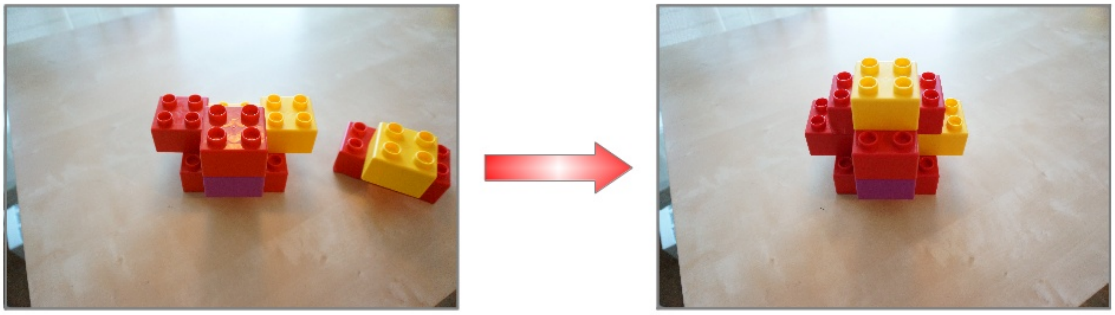


Figure 4.11: *If the Lego block that is moved behind the high structure was smaller, it would be occluded and the tracking would fail.*

structure in the front, and the other consists of a 2x4 brick and a 2x2 brick. In this case the Kinect is able to track the movement of the smaller block of Legos. It is easy to see, however, that if the smaller structure was not wide and high enough, it would be occluded behind the bigger structure, and the tracking would fail. Thus, when building a large structure, one must carefully consider where to place the Legos.

The tracking process in one assembly operation failed partially during the demonstration of this assembly task. As a result of the failed tracking, one block was so ill-fitted, that there is one link missing between nodes. Hence, there is an error in the learned adjacency matrix of the original assembly product. This error is difficult to find in the adjacency matrix, and is not really seen even in the simulated model of the structure: The ill-fitted Lego block is just a bit crooked. However, this error occurs in a later stage of the learned assembly, and thus it does not influence the inexact replications significantly. In a later stage of the assembly such a block is usually not available — as needed for the erroneous exact replication — and inexact replication figures out another, error-free, configuration. Note that this error would be repeated in exact replications. However, if the misplacement is negligible, the error is generally harmless.

In addition to the aforementioned error, there were also several other inaccurate tracking operations of blocks in this experiment. These blocks were not significantly ill-fitted, that is, no links were missing due to the inaccurate tracking, and hence they did not affect the adjacency matrices of the assembly products.

This time the Laplacian measure managed to construct a fairly wide structure when only 2x2 bricks were used (**a** in Figure 4.12). Also, both measures produced rather wide configurations when many 2x4 bricks were used (images **c** – **e** and **h** – **j** in Figure 4.12), although the resemblance to the original configuration is obscure.

Seventh Inexact Experiment

This final experiment contains the largest structure of all experiments: The originally observed structure comprises of three 2x4 bricks and six 2x2 bricks. In this assembly

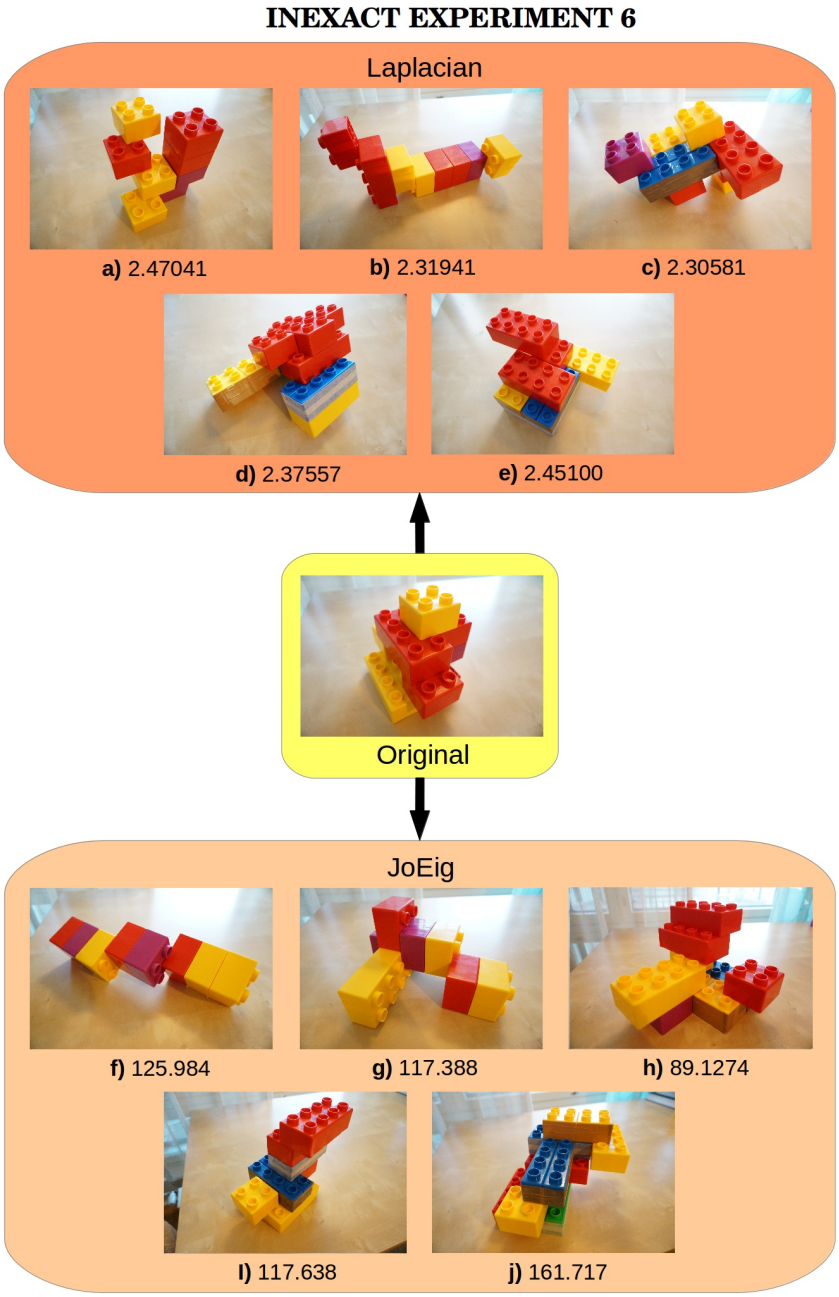


Figure 4.12: *There exists a base of two 2x4 bricks as the undermost bricks in the original configuration, although it is not clearly visible. The original structure contains three 2x4 bricks and five 2x2 bricks. The structures* ***a*** *-* ***e*** *are replicated using the Laplacian measure, and the structures in* ***f*** *-* ***j*** *are replicated using the JoEig measure. The number of 2x2 and 2x4 Lego bricks changes in the replications as explained in the caption of Figure 4.9. The values represent the dissimilarity between the replicated structures and the originally observed structure.*

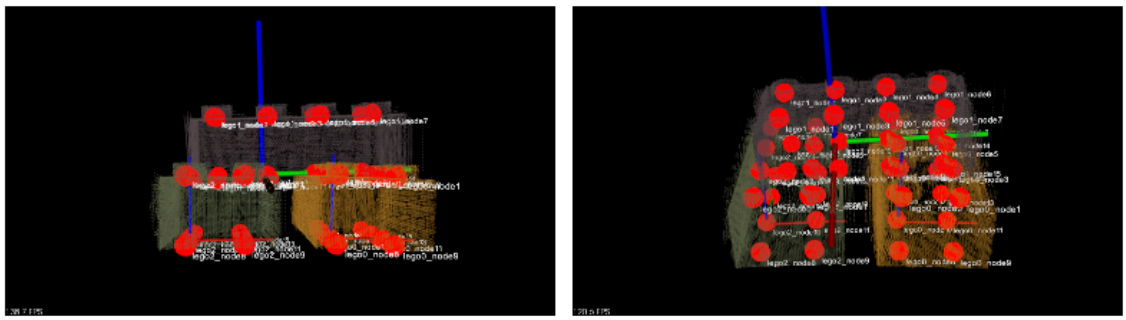


Figure 4.13: *In the second assembly operation of the seventh inexact experiment a block is seriously ill-fitted. A block made of the yellow and blue (the one on top) bricks is incorrectly connected to the green Lego brick. This might have transpired because the larger block was ill-fitted in the tracking procedure, or the green Lego moved during the assembly operation, or both. These images show how the bottom nodes (the red spheres) of the blue brick are not connected to the upper nodes of the green brick. In other words, there are four disconnected spheres on the top of the green brick in the left image. If the nodes were connected, there would be only two spheres (or two separate red blobs) visible, as the nodes of the blue brick would coincide with the nodes of the green brick.*

task the human teacher has first created multiple subassemblies of two or three Lego bricks. Then, he combines these into the final assembly product. However, as in the previous experiment, this experiment also contains an error due to an ill-fitted Lego brick during the demonstration. This time the error occurs already in the second assembly operation, when all three 2x4 bricks are combined. Figure 4.13 presents the models and nodes of these ill-fitted bricks. Clearly the green 2x4 brick (on the bottom left in the images) is misplaced, and accordingly, the nodes are also misplaced. This results in missing links between nodes in the graph and adjacency matrix.

Unlike in the previous experiment, this time the error occurs early in the demonstration. This poses a serious problem, since the faulty block remains in the replication always when we have three 2x4 bricks available. Consequently, in this experiment, we show the results only for those sets of Legos where the 2x4 bricks are replaced with 2x2 bricks. If we replace any of the 2x2 bricks with a 2x4 brick, the error remains in the structure, and there is a high possibility that a brick will be inserted into the misaligned nodes. This results in a physically impossible configuration, where a brick is inside another Lego brick.

Furthermore, in this and the previous experiment the inexact replications are affected by the erroneous adjacency matrices. At each stage of the assembly task we use the adjacency matrices to figure out the best move. Since there are links missing in the learned adjacency matrices, the inexact replication uses these faulty adjacency matrices to find the best configuration.

Figure 4.14 presents the results for this experiment. The lack of 2x4 bricks seems to force the Laplacian dissimilarity measures to build tower-like structures. The

configurations produced by JoEig, on the other hand, are fairly wide. All the inexact replications constructed using the JoEig measure are unstable, whereas **c** is stable, and **b** holds for a few seconds before the purple 2x2 brick in the front falls off. This is yet another flaw in the system: From time to time, the algorithm suggests a configuration where a Lego is not supported from below, and hence will typically fall at some point.

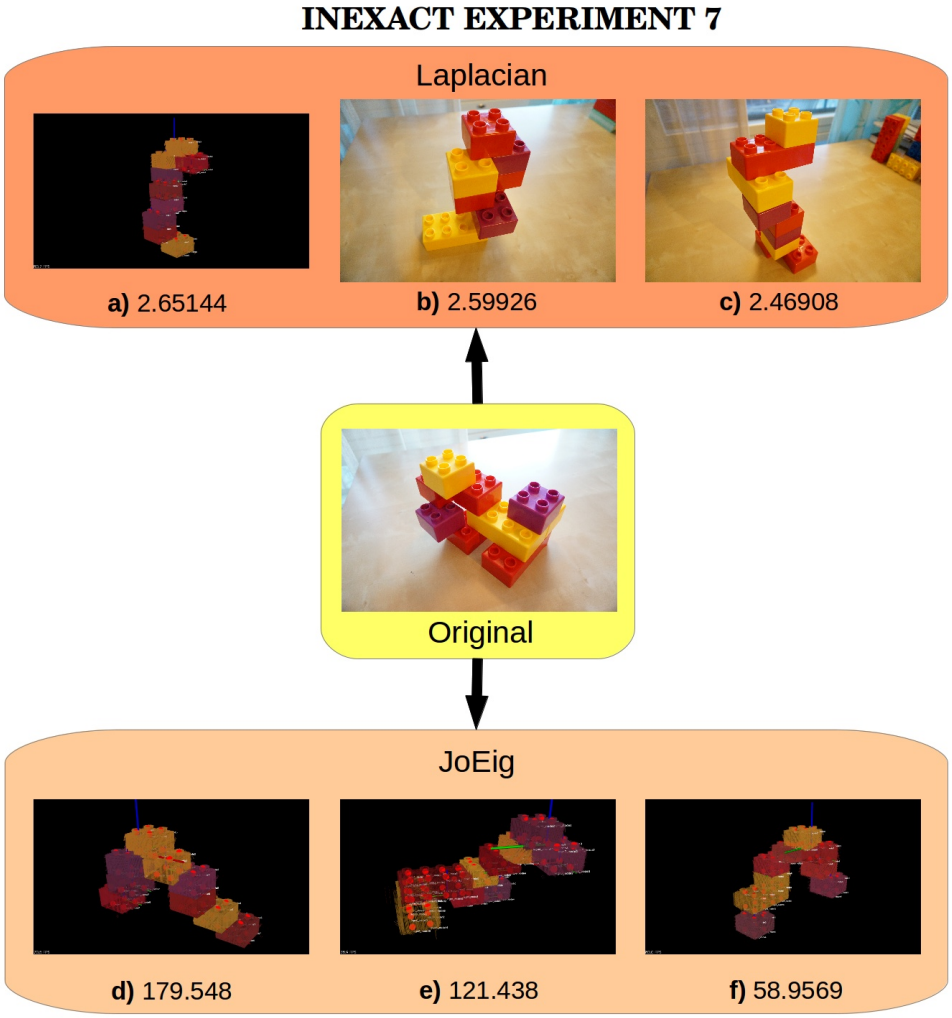


Figure 4.14: *The original structure is comprised of three 2x4 bricks and six 2x2 bricks. The images* **a** *-* **e** *present the structures built using the Laplacian measure, and the images* **f** *-* **j** *present the structures built using the JoEig measure. The number of 2x2 and 2x4 bricks change in the replications as explained in the caption of Figure 4.9. The values represent the dissimilarity between the replicated structures and the originally learned structure.*

4.3 Greedy Construction

In these experiments the robot is given only the graph of the final assembly product, and its task is to build a similar Lego structure. Now the robot does not have any knowledge of the intermediate assembly operations, and thus there is no learning involved in these experiments. The robot simply uses inexact graph matching at each step to compare the coalescence of two available graphs to the desired final graph. As in the inexact replication experiments, the robot simulates all possible configurations, and chooses the one that maximizes the similarity between the current and the target graph. Therefore, there is no planning involved, and we choose greedily the best next step.

The experiments are conducted as follows. We use the last four experiments of Section 4.2 as the target assembly products. Then we provide only the graph of the final product to the robot, and omit all the other assembly operations. The robot begins the construction by examining what kind of graphs it has available to it, initially only 2x2 and 2x4 bricks. Then it goes through all possible configurations involving these Legos, and chooses the one that maximizes the similarity between the currently available graphs and the final graph. Once such a configuration is found, the robot performs this assembly operation and joins the chosen Legos to achieve the selected configuration. Next the robot starts from the beginning and examines what types of graphs are available, and again goes through all possible configurations. Initially the robot is given the same set of Legos that was used in the original demonstration, and it will repeat the described cycle until all Legos are connected into each other.

Figure 4.15 displays the results for this greedy construction, and compares them to the originally observed assembly products. With both measures, the first experiment is relatively well constructed. The next two experiments are not so well managed by the Laplacian similarity measure, but the JoEig measure handles these replications surprisingly well. The cost of these replications is nearly zero.[17] Curiously, the constructed structure in experiment 2 is not equal to the original structure — although the graphs are nearly the same. This reveals one shortcoming in the graph spectra based dissimilarity measures: Two graphs may be isospectral, that is, they have equal multisets of eigenvalues, even when the graphs are not isomorphic. Thus, these dissimilarity measures may erroneously find two different structures similar. In experiment 3, however, JoEig can produce the same structure as originally observed.

In experiments 4 – 7 the similarities between the constructed and original structures are more difficult to perceive. The costs in all experiments, however, are significantly lower than in the inexactly replicated experiments. Although, in these experiments the set of assembly parts is complete, unlike in the inexact replication experiments.

[17]The cost is not exactly zero, because the links are not between the exactly same nodes: One of the Legos is rotated around its z-axis as compared to the originally observed demonstration. Since the links are in different positions in the adjacency matrices, the eigenvectors are different, which results in a non-zero cost. However, the eigenvalues of the original structure and the replicated structure are the same, although the eigenvectors are not.

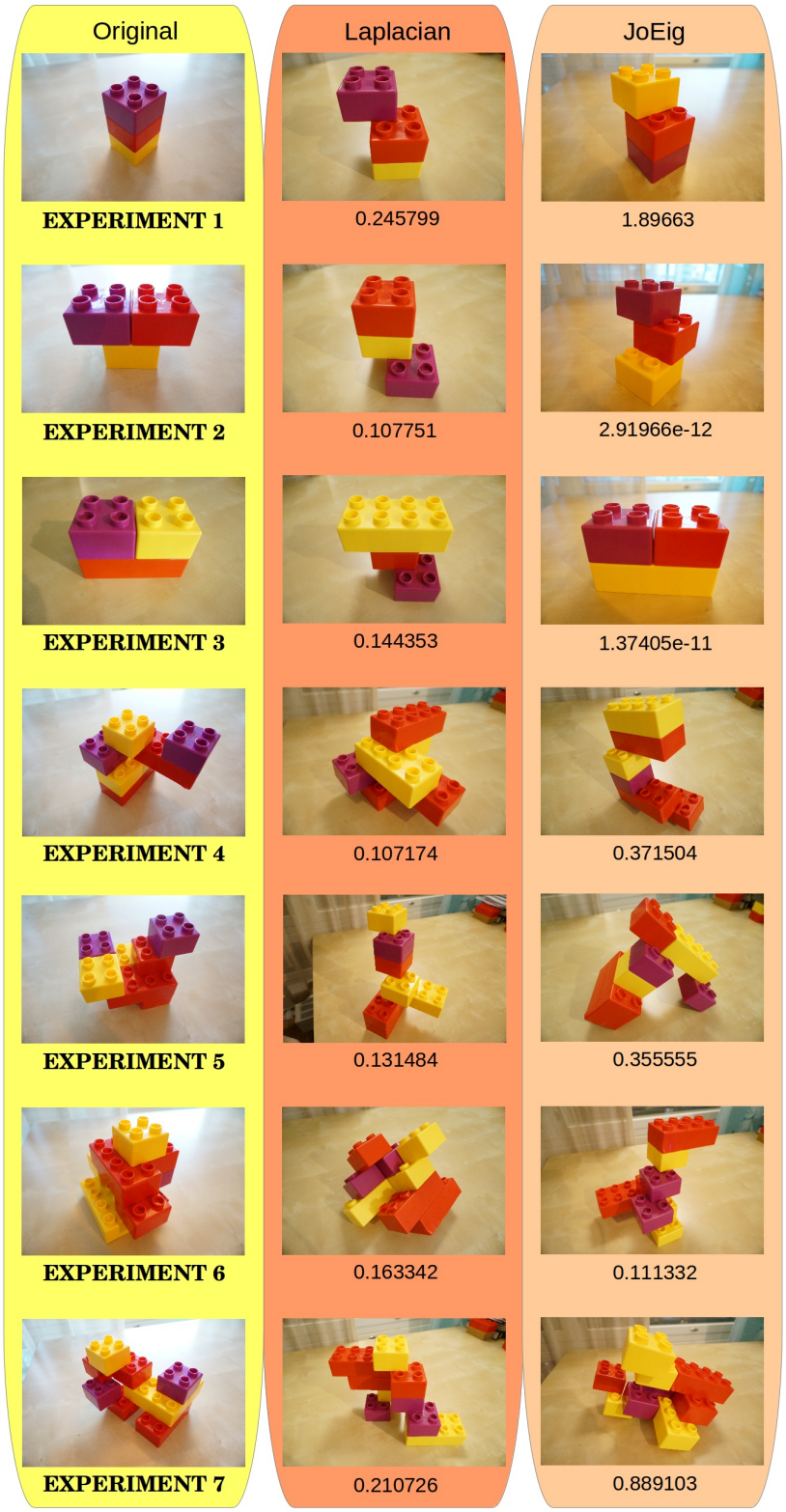


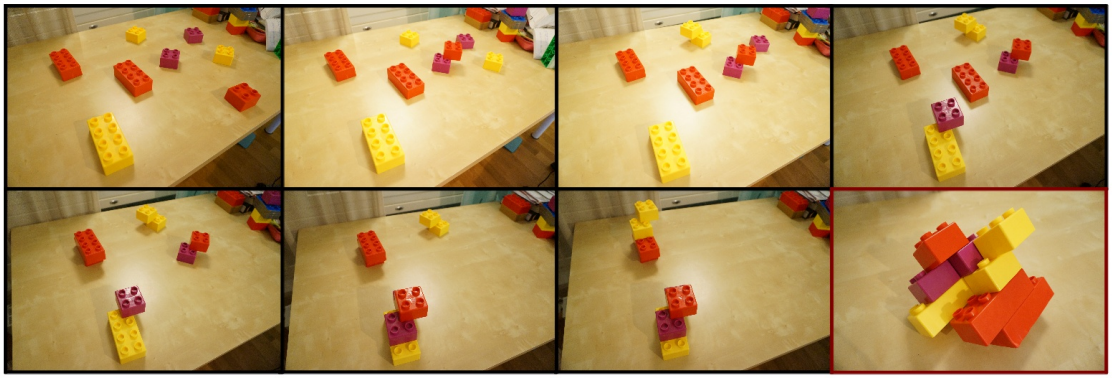
Figure 4.15: *Originally learned assembly products are on the yellow background, greedy constructed assemblies are on the reddish backgrounds. Each row corresponds to one of the experiments. The values represent the dissimilarity between the replicated structures and the originally observed structure.*

Clearly both measures find these greedily constructed structures extremely similar to the original ones, even though to the human eye this similarity is not obvious.

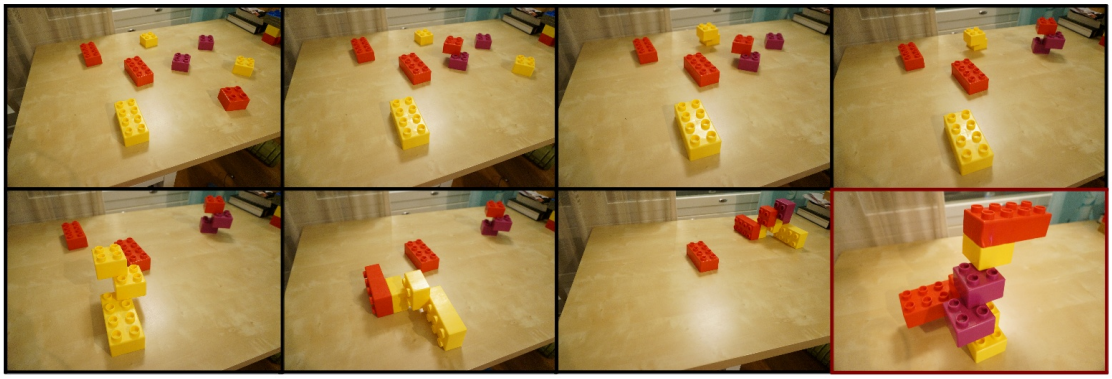
The Laplacian and JoEig measures behave differently in the greedy construction experiments. The cost, as measured by the Laplacian dissimilarity measure, actually increases at each stage in the construction. This measure always finds a single 2x2 or 2x4 brick more similar to the final graph than any combination of these bricks. This exhibits an expected problem of the greediness: When we are interested in optimizing only the next best move (minimizing the cost), we might get stuck in a local minimum. The JoEig measure, on the other hand, generally finds a combination of Legos more similar to the desired graph, and does not get stuck in this minimum. The reasons for this discrepancy are discussed in Section 5.

Because of the local minimum, the Laplacian similarity measure always begins the construction in the same way, regardless of the final assembly product. Figure 4.16a demonstrates a typical greedy construction with the Laplacian measure. First, the algorithm combines a 2x2 brick with another 2x2 brick such that they are connected only through the corner. This operation has the lowest cost, that is, the block formed by this operation resembles most the desired graph of the final assembly product. The cost of adding another brick into this combination is still higher than reproducing the same operation with another set of 2x2 bricks. This operation is repeated as many times as there are 2x2 bricks available. When there are no more 2x2 bricks, the measure joins single 2x4 bricks. When all single blocks are connected, the algorithm begins to join the small subassemblies together. Finally the large two subassemblies are joined together to form the final assembly product.

Since the JoEig measure does not generally get stuck in the minimum, its construction looks distinctly different. Figure 4.16b displays the greedy construction of experiment 6 using the JoEig measure. In this particular experiment, however, the measure first starts to build two distinct blocks of Legos, similarly as the Laplacian measure. However, it then resumes adding the rest of the single bricks into these existing structures. Finally the two subassemblies are combined.



(a)



(b)

Figure 4.16: *Greedy construction of experiment 6 using the* ***(a)*** *Laplacian and* ***(b)*** *JoEig dissimilarity measures. Each individual assembly operation is depicted here: The order is from left to right, top to bottom. The last image, in red brackets, shows the final assembly product — as seen already in Figure 4.15. In* ***(b)*** *the structure was not stable, hence the subassemblies are on their sides in the last images.*

5 Discussion

In this section we will first address a few issues within the learning phase of our system. Then, we will discuss issues within the replication phase, as well as analyse the results from the previous section in depth.

5.1 Learning Phase

Two of the most vulnerable parts of our system are the recognition and tracking processes, which are utilized in the learning phase of an assembly task. In order to perform the experiments, we had to factor in several critical assumptions due to these processes. For example, the Lego bricks had to be in an upright position, far enough from each other, and clearly visible during the learning phase — although, some occlusion was permitted. However, even though these assumptions were followed, we still encountered complications in the experiments. In experiments containing several Legos, such as inexact replication experiments 4 – 7, the complications often interrupted the learning phase. Therefore, we had to repeat these experiments multiple times to learn the task correctly — and still some errors remained in experiments 6 and 7. Albeit, these two experiments were partly chosen *because* they illustrate some of the problems.

Most of the complications occurred during the recognition and tracking processes. First of all, the individual Lego bricks were not always identified correctly in the beginning of the demonstrations. A 2x2 Lego brick might have been recognized as a 2x4 Lego brick, and vice versa. One factor that contributes into this incorrect identification might be the incomplete training of the recognizer: The recognizer was taught to identify the Legos with a limited set of partial views (see Section 3.2.3 for a description of how the recognizer functions). Thus, when a Lego is in a "favourable" pose, a partial view of a wrong type of a Lego is matched to the observed cluster, which results in an incorrect recognition. Secondly, sometimes a wrong partial view of the correct type of a Lego is matched to the observed cluster, which results in an inaccurately estimated pose. This is, however, often corrected by the ICP algorithm, which is run after the initial recognition. Moreover, one specific problem occurred frequently in the experiments: A 2x4 Lego brick was extremely difficult to recognize when its long side "points outwards" of the Kinect. Frequently the Lego was not recognized at all in this pose. The recognized 2x4 brick was also often rotated 90 degrees into the wrong direction, such that its z-axis, which should point into the same direction as the tabletop's surface normal, pointed into a direction that was perpendicular to the tabletop's surface normal.

The 2x4 brick caused problems with the tracking process as well. During the tracking, the model of the brick had a tendency to rotate 90 degrees around its z-axis when the long side points outwards of the Kinect. However, a major problem in the tracking process concerns the smaller 2x2 brick. Occasionally, when a 2x2 brick is placed either on top of another 2x2 brick or a 2x4 brick, the model which we are tracking

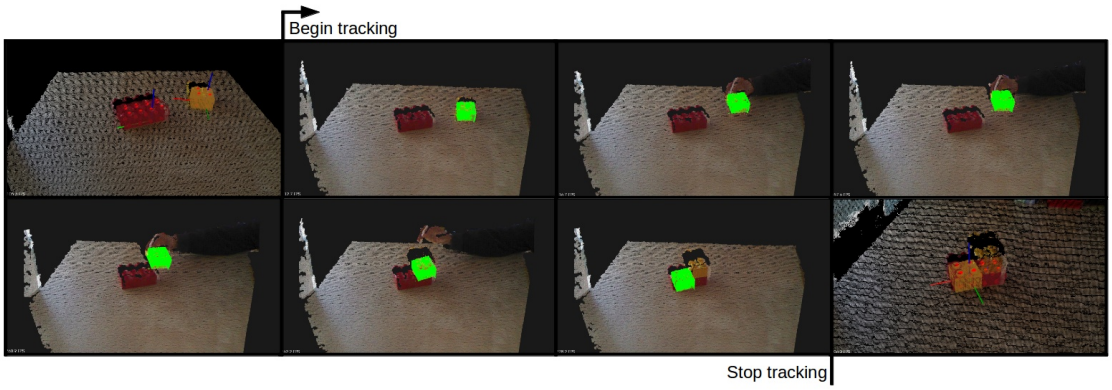


Figure 5.1: *A series of images illustrate how the tracking jumps from one brick to another. Read images from left to right, top to bottom. The first image shows the original situation where we have a 2x2 brick and a 2x4 brick. Then, we begin to track the 2x2 brick, and the human teacher moves it on top of the 2x4 brick. After a while the tracking might jump onto the 2x4 brick. The last image depicts the situation where ICP has been applied to estimate the pose of the manipulated Lego. Now, clearly, the 2x2 brick is incorrectly estimated to reside inside the 2x4 brick.*

"jumps" from the brick that was moved into the brick that was not moved. Figure 5.1 illustrates how this jump transpires. The jump occurs because some of the particles of the particle filter move into the area of the 2x4 brick, and gain more weight than the remaining particles in the area of the 2x2 brick. The teacher can, however, rectify the situation by using a fickle and unreliable technique, where he 1) places his arm between the Kinect and the Legos such that the Kinect cannot get depth values for the pixels in the arm's shadow, and 2) moves his arm — and the shadow with no depth values — on top of the misplaced particles, thus pushing the particles back onto the correct brick. The particles, and the model which we are tracking, then jump back to the area of the correct brick.

Sometimes, however, the pose of the tracked block is estimated poorly even if the tracking is successful. This usually happens when the ICP, which we employ after the tracking, fails. After a failed ICP, the model of the Lego is ill-fitted onto the observed cluster of the relocated Lego. This ill-fitting results in crooked models of Legos inside the simulated structures (see Figures 5.2 and 4.13). In the worst case an ill-fitted block is crooked enough to cause missing links inside the graph structure, as happened in inexact experiments 6 and 7[18].

At least some of the aforementioned issues with recognition and tracking could be averted by using other techniques. Both the recognition and tracking methods were chosen because they were easy to use: They required only a model of the object which is to be recognized/tracked, and nothing else. Other recognition and tracking methods that rely on detecting markers that are attached to the objects, such as [81],

[18]Although, the ill-fitting in experiment 7 was probably due to an erroneous movement of a wrong Lego brick in the tracking process, and not failed ICP.

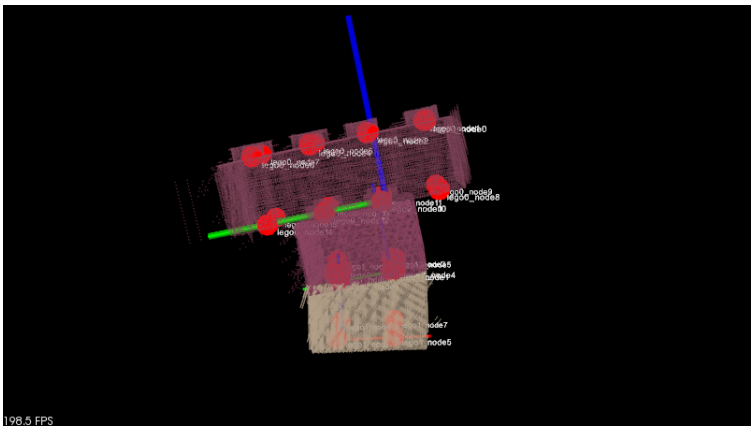


Figure 5.2: *Here a block consisting of a 2x2 brick and a 2x4 brick is ill-fitted in the learning phase. This simple structure would still produce a correct graph, because the nodes between the two 2x2 bricks are connected. However, if the teacher put a Lego brick on top of the 2x4 brick, these bricks would be substantially misaligned. This would result in missing links within the graph structure.*

could possibly have worked more reliably. Also, the particle filter tracker provided by PCL is based on an old, and possibly outdated, method [82], and newer improved methods might be available.

5.2 Replication Phase

We will first go through the results and issues of the inexact replication experiments. We also attempt to analyse the possible sources of those issues. Then, we will process the results and issues of greedy construction experiments in a similar manner.

5.2.1 Inexact Replication Experiments

There were two critical restrictions in the inexact replication experiments: The robot must use the same number of Legos as was used in the observed task, and choose greedily the assembly parts before replicating an operation. Nevertheless, in certain situations the robot could produce more similar structures if it used less or more Legos than the human teacher did. For example, if a 2x4 brick is used to replace two adjacent 2x2 bricks, the robot could skip one operation where the other 2x2 brick is placed next to the first one. Or vice versa, when a 2x4 brick is replaced with a 2x2 brick, the robot would need one more operation — and an extra 2x2 brick — to fully compensate for the 2x4 brick. The latter situation might require some planning also, otherwise the first 2x2 brick might be placed such that there is no room to place the second brick. There was one reason why we preferred to use the same number of operations: the problem of a local minimum. We will discuss this problem in detail in Section 5.2.2. Essentially, it means that the dissimilarity measures are likely

to find a small structure, even a single brick, more similar to the desired assembly product than a larger structure. Hence, the robot would probably terminate the task after a few or even zero assembly operations. By replicating the same number of operations as the teacher did, the robot is guaranteed to build larger, and hopefully more similar, structures.

Secondly, instead of choosing greedily the assembly parts, the robot could go through all possible combinations of available Legos, and find the structure that was most similar to the desired post-operation product. In other words, the robot could have used the same approach as in greedy construction experiments, except the desired configuration would be the post-operation product, instead of the final assembly product. However, this approach would have required a lot of computational power, since the robot must try all possible configurations: First put one block of Legos on top of or under another block of Legos, and try all possible configurations in different rotations, and so on. The only downside in this approach is that it slows down the replication procedure. Nevertheless, by comparing all plausible configurations in each operation, the measures would have probably produced more similar structures. After all, the end product of an assembly operation is more important than which parts were used — although, this might not be true in other tasks with different types of assembly parts.

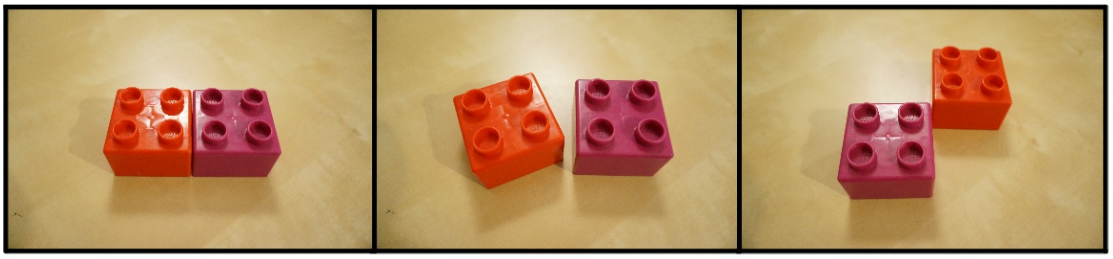
The problem of ill-fitted blocks reveal a serious shortcoming in our proposed method: If a demonstration is faulty, also the replications where the defective assembly operation is performed are faulty. If a block is so ill-fitted, that some links are missing, the consequences may be serious in the inexact replications. The significance of the poorly fitted block depends on whether it happens early or late in the assembly. In the sixth inexact experiment the missing link in the graph structure was not serious, but in the seventh inexact experiment the missing links prevented us from performing some of the replications. One solution is to incorporate some sort of *sanity checks* after each assembly operation. For instance, the program could check that the models of the Lego bricks are not inside each other, or that the coordinate frames of the models are properly aligned, and not crooked. Currently the program merely checks that each node is connected at most to one node in another brick. In other words, the program checks that one node is not connected to two or more Lego bricks, as that would be an physically impossible situation.

On the other hand, multiple structures in the inexact replications were unstable, and some sort of stability checks would be more than welcome. Out of the total 34 replicated structures in the inexact replication and greedy construction experiments, 14 structures built using the Laplacian similarity measure were unstable, and 17 structures built using the JoEig similarity measure were unstable.[19] Thus, 41.2% of the Laplacian and 50% of the JoEig structures were unstable. Consequently, some sort of stability checks or metrics (such as [17] presents) are required to improve the quality of inexactly reproduced structures. A more arduous solution would be to

[19]Here we count a replication as unstable if the final assembly or any subassembly product is unstable, or if the replication contains an impossible assembly operation (see Figure 4.8).

improve the system such that it either explicitly or implicitly learns which assemblies are stable, and which are not.

The issue of placing two Legos side-by-side on the tabletop was encountered in two inexact replication experiments. While this is not a problem in the exact replication — as we just use the known transformation matrix between the Legos to move them side-by-side — the inexact replication scheme cannot place two blocks side-by-side on the tabletop. In inexact replications the robot has not been taught explicitly where to put a Lego block, and thus it does not have a transformation matrix explaining the relation between the two chosen blocks of Legos. As is explained in Section 3.3, in these replications we employ inexact graph matching to find the best configuration between two blocks of Legos. More specifically, using inexact graph matching our algorithms provide the transformation matrix that accomplishes this configuration: First the dissimilarity measures are used to compare the adjacency matrices of different configurations to uncover the best one. Then, our system produces the transformation matrix along with the adjacency matrix of the chosen configuration. However, when the Legos are put side-by-side on the tabletop, they do not share any connections. As long as the Legos are close enough to each other, their relative locations can be almost anything, and they still produce the precisely same adjacency matrix. For example, in Figure 5.3 all three different configurations produce the exactly same adjacency matrix. Since there is an infinite set of transformation



(a)

$$\boldsymbol{A} = \begin{bmatrix}
0&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\
1&0&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\
1&1&0&1&1&1&1&1&0&0&0&0&0&0&0&0\\
1&1&1&0&1&1&1&1&0&0&0&0&0&0&0&0\\
1&1&1&1&0&1&1&1&0&0&0&0&0&0&0&0\\
1&1&1&1&1&0&1&1&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&0&1&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&0&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&0&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&0&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&0&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&0&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&0
\end{bmatrix}$$

(b)

Figure 5.3: *All the configurations in* ***(a)*** *produce the same graph, and therefore the same adjacency matrix, which is shown in* ***(b)****. As long as they are close enough to each other, they are regarded as one assembly product and one graph.*

Table 5.1: *$\delta_{Laplacian}$ is the cost for the Laplacian measure, and δ_{JoEig} is the cost for the JoEig measure. In both measures the cost increases as the number of Legos grow in the experiments (and thus in the final assembly product). These values represent the average cost over the inexact replication experiments.*

	Number of Legos in an Experiment				
	3	6	7	8	9
$\delta_{Laplacian}$	0.9724	1.9155	2.1365	2.3844	2.5733
δ_{JoEig}	62.4708	101.4826	97.5069	122.3709	119.9810

matrices between two side-by-side Lego blocks, we cannot know which one is the correct one. In practice this could be circumvented by using heuristics. For example, *"Put a block next to another block on the tabletop such that a maximum number of unoccupied nodes are adjacent, use it as the side-by-side configuration, and compute its cost."* — but these heuristics are not implemented in this thesis. Therefore inexact replications that require side-by-side configurations will fail in this respect, and the program will instead place the Legos on top of each other.

In several experiments both dissimilarity measures produced a tall, tower-like structure when mainly 2x2 bricks were used. The inability to generate side-by-side configurations on the tabletop partially explains this behaviour. On the other hand, the small and narrow bricks also hamper the construction of wide structures, as each added 2x2 brick cannot broaden the structure significantly. However, an exception to this was seen in inexact replication experiment 7, where the JoEig measure managed to generate rather wide structures with 2x2 bricks.

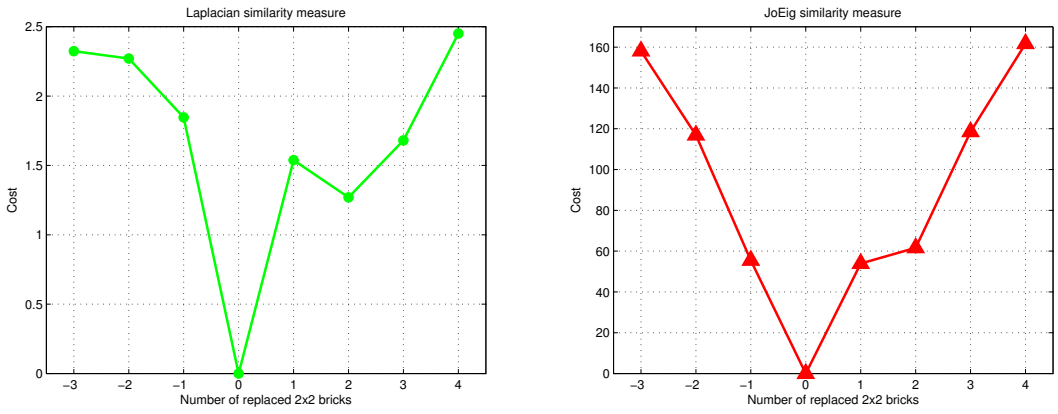
All in all, both dissimilarity measures produced fairly similar structures in the inexact replication experiments. Some of these structures resembled the originally learned assembly products, most of them did not. The most notable difference between the measures is that the Laplacian costs were significantly smaller. As Table 5.1 shows, both dissimilarity measures increase when the number of Legos in the final assembly product grows. As explained in Section 2.3.1, we chose to use $K = M$ eigenvalues when computing the JoEig dissimilarity measure, and $K = M - 2$ first eigenvalues when using the Laplacian dissimilarity measure, where M is the size of the smaller graph. Hence, when the size of the graph increases, also the number of eigenvalues grow. Remember that in the Laplacian dissimilarity measure the dissimilarity is computed as the Euclidean distance between the vectors consisting of the graphs' eigenvalues. As the dimensionality of these "eigenvalue vectors" grows, so does the distance between them, because, generally, the eigenvalues are not equal. Therefore, the addition of Legos into the assembly product increases the cost. Presumably something similar happens with the JoEig measure, but it is more difficult to speculate because the eigenvalues of the graphs are mapped into the joint eigenspace.

Moreover, in both measures the cost increased when more Lego bricks were replaced. This is not surprising, because when more and more bricks are replaced, the further away the replicated graph drifts from the graph of the learned assembly product. Figure 5.4 displays how the cost increases as a function of replaced bricks. In the x-axis the negative values indicate how many 2x4 bricks are replaced with a 2x2 brick, and the positive values specify how many 2x2 bricks are replaced with a 2x4 brick. In $x = 0$ the cost is zero, because no bricks are replaced, and the set of Legos is the same one which was observed in demonstrations. The values displayed in the figure are averaged over all inexact replication experiments. By visually examining the inexactly replicated structures in the inexact experiments, it is obvious that the dissimilarity measures cannot generate similar structures when several bricks are replaced: At least not in the sense how a human typically perceives similarity. Instead, the replicated structures look significantly different when compared to the original assembly products.

An often encountered behaviour with both measures was the preference for a configuration where the Lego bricks are connected through only one corner node. A good example of this is seen in Figure 4.6, configuration **d)**. The adjacency matrix of the two 2x2 bricks is

$$\boldsymbol{A} = \begin{bmatrix}
0&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\
1&0&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\
1&1&0&1&1&1&1&1&0&0&0&0&0&0&0&0\\
1&1&1&0&1&1&1&1&0&0&0&0&0&0&0&0\\
1&1&1&1&0&1&1&1&0&0&0&0&0&0&0&0\\
1&1&1&1&1&0&1&1&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&0&1&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&0&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&0&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&0&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&0&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&0&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&0
\end{bmatrix}. \tag{5.1}$$

The two clusters of ones describe the links between the nodes inside the 2x2 bricks. Each 2x2 brick contains 8 nodes, and all of these nodes are connected to each other. In this configuration the link joining the two bricks is situated near the center of the adjacency matrix. Curiously, a similar matrix was encountered in multiple occurrences of the "corner configuration." Note that, in this case, there are 15 other adjacency matrices which depict an equal corner configuration — any corner node of the bottom Lego could be connected to any corner node of the top Lego. Undoubtedly, the measures preferred an adjacency matrix where the link is located close to the "masses" of ones. This behaviour indicates that the dissimilarity measures might focus on irrelevant characteristics of the adjacency matrices. Instead of favoring adjacency matrices where the links reside near the masses of ones, the measure should prefer link locations that represent geometrically or topologically similar graphs. On the other hand, the order of columns and rows does not affect the eigenvalues of the adjacency matrices. Thus, the locations of the links should not affect the Laplacian dissimilarity measure, and also have only minimal effect on the JoEig measures. It remains unclear what is the underlying cause for the recurring occurrence of the the corner configuration.



(a) *(b)*

Figure 5.4: *In both measures, the cost increases as a function of replaced bricks. The negative values in the x-axis indicate how many 2x4 bricks are replaced with a 2x2 brick, as compared to the set of Legos which the teacher used in the demonstration. Likewise, the positive values specify how many 2x2 bricks are replaced with a 2x4 brick.* ***(a)*** *The Laplacian similarity measure, and* ***(b)*** *the JoEig similarity measure.*

Moreover, the adjacency matrices of assembly products comprised of many bricks are really large. However, since a node can be connected to only one other node in another brick, there is a maximum of 8 or 16 ones in a column/row of the adjacency matrix. For instance, if a structure consists of ten 2x2 bricks, and all the nodes are occupied, there are eight ones and 72 zeros in each column/row[20]. Therefore, both the adjacency and Laplacian matrices are quite sparse in a large graph, and the dissimilarity measures used in this thesis may not exploit the characteristics of these matrices optimally. Or, alternatively, there might be a more suitable representation for the graphs than the adjacency and Laplacian matrices.

It seems as the dissimilarity measures consider topological rather than geometrical similarity. Topological similarity means that the characteristics of the graphs are similar — there are links between the same nodes in certain positions. The graph representation we use accentuates this topological similarity, as the nodes and links do not encode strictly geometrical information. For example, imagine two 2x2 bricks in the corner configuration. As the bricks are connected through only one corner, the top brick is free to rotate to some degree. However, when the top brick is rotated and the geometric relation is changed, the produced graph is still the same. Another example is provided in Figure 5.5, where both structures produce the exactly same graph. This is because the graph of a single 2x2 brick is completely symmetric — you may rotate it anyway you want and it remains unchanged. We could encode some geometric relations into the graph by, for instance, using distances between the nodes as links' weights. In this case the graph of a single brick would not be totally symmetric, and the two structures in Figure 5.5 would not have the same graphs.

[20]To be exact, such a structure is impossible. The undermost and uppermost nodes of any structure are unoccupied, unless one uses curved Legos to create a ring.

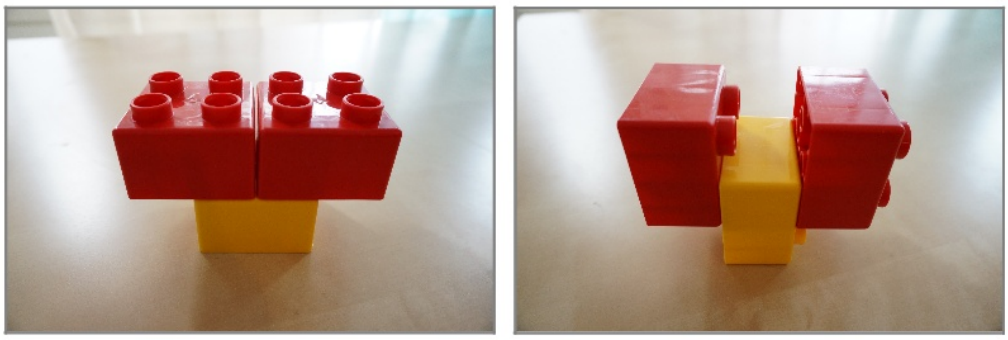


Figure 5.5: *Both structures produce the exactly same graphs. The graph representation encodes topological information rather than geometrical.*

5.2.2 Greedy Construction Experiments

The previously mentioned corner configuration was a familiar sight in the greedy construction experiments as well. With the Laplacian measure nearly all experiments began by forming pairs of single bricks, such that they were connected from the corners — see Figure 4.16a, for example. The JoEig measure, on the other hand, seemed to generate a more diverse set of configurations. Hence, the adjacency matrix of the final assembly product had a larger effect on the assembly operations when the JoEig measure was used. This suggests that the JoEig measure considers the characteristics of the final assembly product better than the Laplacian measure — at least in the first assembly operations.

The Laplacian measure prefers to form pairs of the single bricks because it gets stuck in a local minimum. The measure finds a single 2x2 or 2x4 brick more similar to a given assembly product, than any combination of the available bricks. But, since we force the greedy construction to use all of the given bricks for the assembly task, the program executes an assembly operation with the lowest cost. The reason why it gets stuck in a local minimum relates to the same reason why the dissimilarity measure increases when the number of Legos grows: The vectors consisting of eigenvalues are higher-dimensional when configurations with more bricks are considered. For example, when we are performing the first assembly operation in Figure 4.16a, the program considers which configuration has the smallest cost: 1) A configuration of two 2x2 bricks, 2) a configuration of two 2x4 bricks, or 3) a configuration of one 2x2 and one 2x4 brick. In 1) the size of the resulting adjacency matrix is 16x16, in 2) 32x32, and in 3) 24x24. Thus, the size of the eigenvalue vector is $K = M - 2$, that is, 1) $K = 14$, 2) $K = 30$, and 3) $K = 22$. As was already concluded, the distance between the eigenvalue vectors increase as their dimensions increase. Therefore, the Laplacian measure chooses the configuration with two 2x2 bricks, as the distance between two 14-dimensional vectors is smaller than the distance between 22- or 30-dimensional vectors.[21] Consequently, the Laplacian dissimilarity measure is better

[21]Remember that only $K = M - 2$ eigenvalues are considered from the larger graph of the final assembly product as well.

suited to the inexact replication experiments, where we have knowledge of how the assembly product should look like after each assembly operation. Thus, the sizes of the inexactly replicated and desired graphs are more alike.

One solution to improve the greedy construction would be to segment the observed adjacency matrix of the final assembly product into smaller chunks. One could, for example, consider the *edge connectivity* of the graph to find the "weakest links" in the structure. By removing these links, the structure would be segmented into smaller disconnected graphs, and these could serve as a initial target graphs for the greedy construction. Ultimately, the greedy construction would connect these separately constructed graphs by comparing them to the fully connected, original assembly product.

All in all, the incorporation of a planning algorithm into our proposed assembly method would most certainly improve the results in inexact replication and greedy construction experiments. Currently our method focuses on finding the best next assembly operation, that minimizes the cost — and maximizes the similarity — between available Legos and a desired configuration of the Legos. Frequently, however, the next best step may not be the best choice when we consider the following second step or a third step. For example, in inexact replication we always use the exactly same Legos in an assembly operation if they are available. However, in a situation where the assembly begins with a 2x2 brick, it could be beneficial to start with a 2x4 brick to gain more stability. Or one could spare a 2x4 brick in the beginning if its lengthiness would be an advantage later on in the assembly. With planning, the replicated structures in inexact replication experiments would presumably look considerably different — and perhaps the replications would then look more similar to a human also.

A well-known example of an assembly task that requires planning is the Cranfield benchmark [83]. The task constitutes of two base-plates, a separator, and different types of pegs (see Figure 5.6) that must be disassembled and assembled in a correct order. For the method presented in this thesis, the Cranfield benchmark would be partly solvable. The problem is that our method, as presented, cannot perform disassembly operations. The addition of disassembly operations would require the program to recognize which parts are dismantled from the assembly product. However, the addition of such a feature would be possible. Otherwise, our suggested method is able to solve the task, as long as the recognizer is taught to identify the required parts, the models of the parts are manually abstracted into graphs, and a human teacher demonstrates the task.

Ultimately, it is extremely difficult to appraise the similarity between two different structures. This is, presumably, the biggest problem in our method. Typically, the similarity between two objects, or assembly products, is estimated by comparing their shapes, sizes, colors, or functional properties. Especially shape is one of the most important attributes when assessing similarity: In our experiments, we judge similarity by the shape of the structures. It is relatively easy for a human to state whether two shapes resemble each other. Nonetheless, in the case of graphs, other

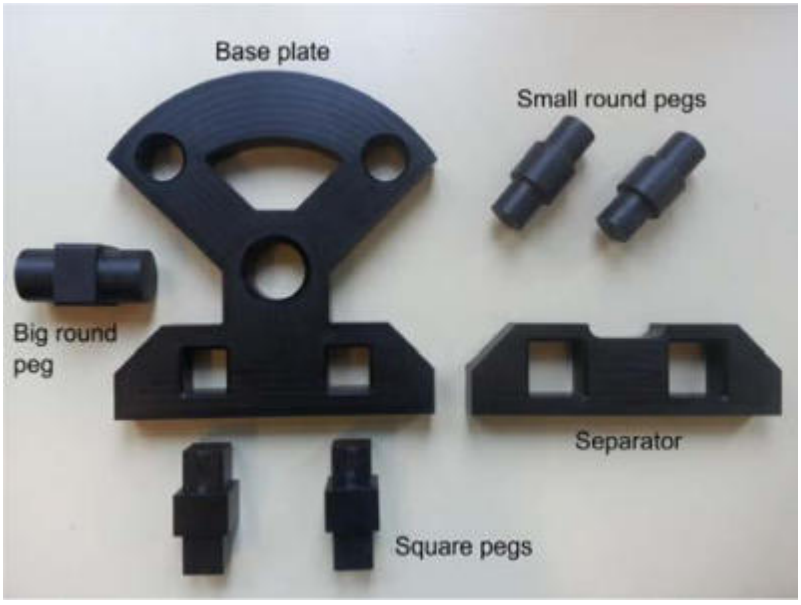


Figure 5.6: *The parts which are used in (one version of) the Cranfield benchmark. The parts must be disassembled and assembled in a correct order. Image from [4].*

properties than shape are usually considered in the similarity assessment. Such properties are, for example, the number of nodes and links, node degree distribution, and clustering coefficient of a graph, and the graph spectrum. In this thesis we have considered the similarity between two structures through two different approaches: By visual scrutiny, and the two similarity measures which relate to the graph spectrum. By examining the results from the experiments, it is difficult to observe a correspondence between the similarity by visual scrutiny, and the similarity measures. In other words, although our method finds the structures that minimize the cost in the inexact replication experiments, they do not resemble the originally observed assembly products (as a human sees it). There are basically two alternatives to overcome this problem: Either we attempt to obtain the correspondence, or mapping, between the similarity apprehension of a human mind and a similarity measure, or then we try to find a similarity measure that already implicitly contains this mapping. The former alternative may be solvable by teaching a robot to distinguish when two structures are similar. The latter option, however, might be difficult to attain — especially if we add, say, functional properties into the notion of similarity.

6 Conclusions

In this thesis we introduced an assembly method in the Learning from Demonstration framework using a graph representation of the assembly parts. The objective of the method is to enable a robot to replicate an assembly task, which is demonstrated by a human teacher. Utilizing the LfD framework to learn assembly tasks is a well-researched area (see, for example, [3, 4, 52]). Our method intends to provide a generalizable and robust framework for learning and repeating various types of tasks. We attempt to learn such a general assembly skill by using a simple and straightforward graph representation of the assembly parts' structures and spatial relations. Furthermore, this representation enables us to compare assembly structures with spectral inexact graph matching methods, and measure the similarity between them. Similarity assessment of assembly structures has been studied before [6, 7], but — to the best of our knowledge — not with spectral inexact graph matching methods. We exploit the similarity assessment to cope with unpredictable changes in the environment, such as replaced assembly parts, or undetected assembly operations. This thesis shows that our method successfully learned to replicate the observed assembly tasks.

An assembly task was learned and replicated in two distinct phases. In the learning phase a robot observed the teacher's demonstration, and formed a database containing the necessary assembly operations. Then, the robot was provided with a set of assembly parts, and attempted to reproduce the task. If the robot was given an incomplete set of parts, it tried to produce a similar kind of structure — since the original assembly product was impossible to attain. We used two dissimilarity measures, the Laplacian and JoEig measures, to evaluate the similarity of assembly products. Both measures used the spectra of the assembly products' graphs to estimate the similarity.

We designed three sets of experiments to test our system, and examine the behaviour of the dissimilarity measures. In these experiments a human teacher demonstrated an assembly task in front of a Kinect using two different types of Lego bricks. The experiments were divided into three sets according to whether the robot replicated the learned assembly operations exactly, or inexactly. In the first set of experiments, the *exact replication experiments*, the robot was provided with the same Lego set which was used in the demonstration. In these experiments, the assembly operations were replicated exactly. Then, in the second set of experiments, the *inexact replication experiments*, the robot was provided with a different set of Legos. In this case, the robot exploited the knowledge of how the assembly product should look like after each operation to complete the task. These experiments required that the assembly operations were replicated inexactly. In the third set of experiments, the *greedy construction experiments*, the robot was only given information of the final assembly product, and not the intermediate operations. In other words, the robot knew what it should build, but was not provided with any kind of hints of how to build it. In these experiments the robot was supplied with the same Lego set as was used in

the demonstration. However, the robot had to replicate the assembly operations inexactly, as it did not have any information of them.

A critical restriction that applied to all types of experiments was related to occlusion of the assembly parts. A part had to be nearly completely visible during the recognition and tracking procedures — a small amount of occlusion was allowed. If a part was severely occluded during tracking, the tracking process failed and the demonstration had to be terminated. Also, the Legos had to be in an upright position when the demonstration began.

The results of the experiments indicated that the assembly method was able to replicate the tasks. The assembly tasks in exact replication experiments were successfully reproduced. Also, when only one or two Lego bricks were replaced in the inexact replication experiments, both dissimilarity measures produced structures that somewhat resembled the learned structures. However, as more bricks were replaced, it became more difficult to recognize the similarity between the demonstrated assembly products and the replicated assembly products. The replicated structures in the greedy construction experiments did not generally resemble the observed structures.

The most notable issues in the experiments concerned the inaccurate pose estimation of assembly parts. Situations where the pose estimation failed after an assembly operation, that is, after a tracking process, were especially problematic. If an assembly part was seriously ill-fitted after tracking, the misplaced block remained in the assembly product. In this event, the only solution was to abort the task, or continue and hope that the erroneous placement would not affect the inexact replications. In exact replications the mistake was repeated, but if the misplacement was minute, the model of the simulated assembly product was sufficiently accurate. Sufficiently accurate means, in this case, that a human could replicate the task according to the simulated assembly model. Occasionally, the inexact replication might even correct the error in the structure, but there was also a high probability that the inexact replication failed due to a severely ill-fitted block. Moreover, the most common reason to abort the learning phase of a task was the "jumping" of the monitored Lego's model into an incorrect Lego during the tracking process.

The inexact replication experiments revealed that the chosen dissimilarity measures behaved in a rather similar manner. When the experiments involved several 2x2 bricks the measures seemed to favor high, tower-like constructions. Albeit, some of this behaviour was explained by the fact that the inexact replication could not place blocks of Legos side-by-side on the tabletop. Therefore, the only direction to build, especially with the narrow 2x2 bricks, was upwards. Moreover, both dissimilarity measures increased when the number of replaced bricks grew, and also when the number of Legos in the experiment grew — although, the latter was slightly less visible in the JoEig measure. The most alarming observation, however, was that roughly half of the inexactly produced structures were unstable. Accordingly, if the experiments were conducted using a real manipulator, these structures would have collapsed. Clearly, in addition to the similarity measure, some sort of a stability measure is required.

The Laplace measure performed poorly in the greedy construction experiments. Essentially, the measure performed inadequately because it got stuck in a local minimum right in the beginning of an experiment. This, on the other hand, was due to the dimensional discrepancy between a small (the initial bricks) and a large (the final assembly product) graph — and of course, the greedy nature of our algorithm. Therefore, the Laplace measure seemed to be more suitable to the inexact replication experiments, where we had knowledge of the intermediate assembly products. However, the JoEig measure seemed to handle the large difference between two graphs' sizes more appropriately, and thus was better suited to the greedy construction experiments.

The aforementioned problems pose several issues for future consideration. First of all, the recognition and tracking processes should be improved to make the learning phase more reliable. Also, to reduce the number of aborted demonstrations, the method should be improved such that it could cope with partly failed demonstrations. Secondly, the similarity measure is a major obstacle. Evidently, the similarity measures based on graphs' spectra do not correspond to the notion of structural similarity that humans have. In future research, one could attempt to devise a similarity measure that aligns better with the interpretation of similarity as humans see it. One could, for example, attempt to teach a robot what type of structures are similar. Thirdly, adding a planner into our method would probably improve the results. Furthermore, as all the experiments were run as simulations, we need to introduce a manipulator into the system. A real-world manipulator poses many new problems that are extremely difficult — if not impossible — to model in the simulations. Lastly, the ultimate goal of our work was to provide a general assembly method, that is able to replicate any assembly task with any kind of assembly parts and operations. We used Lego bricks and pick-and-place operations to prove that the concept works, but this is not sufficient: The system should be tested with different kinds of assembly parts and operations. In our method, the 3D models of the parts and their abstraction into graphs had to be done manually before initiating the learning phase. An automatic procedure to create graphs from observed objects is required to build a more practical method. Also, the current encoding of operations into the database does not allow the learning of other types of operations — for instance, joining parts with a screwing motion. The encoding requires further development to account for different types of operations.

In the future humans will encounter robots more often, and in more diverse surroundings. It is extremely important that the interaction between robots and humans becomes simpler and easier. One way to facilitate the interaction is the Learning from Demonstration paradigm, which mitigates the process of teaching robots new skills. The kind of skill which is required to successfully execute assembly tasks is a relatively complex one. Once a robot is able to learn such a general skill, it can be taught to perform a remarkably diverse set of tasks. Also, the real world is a changing and unpredictable environment. It is of paramount importance that more intelligent robots are developed to work in such surroundings. The intelligence of robots can be improved by augmenting them with the capability to "improvise"

in unanticipated situations. This capability is achieved by enhancing the robots' knowledge of the structural and functional properties of the surrounding objects. By understanding what objects do and what they can be used for, a robot can exploit them in unexpected circumstances — such as in the inexact replication experiments, where some of the assembly parts were replaced.

In conclusion, the method presented in this thesis was capable of reproducing observed assembly tasks, even when supplied with imperfect information of the assembly operations. However, the similarity between the reproduced structures and the learned structures was often elusive. In general, it is difficult to say whether two structures look similar, and how the similarity should be measured.

References

[1] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Survey: Robot programming by demonstration. In *Springer Handbook of Robotics*, editors B. Siciliano and K. Oussama, pages 1371 – 1394. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-23957-4. doi: 10.1007/978-3-540-30301-5_60. URL http://dx.doi.org/10.1007/978-3-540-30301-5_60.

[2] B. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5): 469 – 483, 2009. ISSN 0921-8890. doi: http://dx.doi.org/10.1016/j.robot.2008.10.024. URL http://www.sciencedirect.com/science/article/pii/S0921889008001772.

[3] Y. Wang, R. Xiong, L. Shen, K. Sun, J. Zhang, and L. Qi. Towards learning from demonstration system for parts assembly: A graph based representation for knowledge. In *Proceedings of the 4th IEEE Annual International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 174 – 179. IEEE, June 2014. doi: 10.1109/CYBER.2014.6917456. URL http://dx.doi.org/10.1109/CYBER.2014.6917456.

[4] F. J. Abu-Dakka, B. Nemec, A. Kramberger, A. G. Buch, N. Krüger, and A. Ude. Solving peg-in-hole tasks by human demonstration and exception strategies. *Industrial Robot: An International Journal*, 41(6):575 – 584, 2014. doi: 10.1108/IR-07-2014-0363. URL http://dx.doi.org/10.1108/IR-07-2014-0363.

[5] L. H. de Mello and A. C. Sanderson. AND/OR graph representation of assembly plans. In *Proceedings of the Association for the Advancements of Artificial Intelligence (AAAI)*, editors T. Kehler and S. J. Rosenschein, pages 1113 – 1121. Morgan Kaufmann, 1986. URL http://dblp.uni-trier.de/db/conf/aaai/aaai86-2.html#MelloS86.

[6] M. Ferch, M. Hochsmann, and J. Zhang. Learning cooperative assembly with the graph representation of a state-action space. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 990 – 995. IEEE, 2002. doi: 10.1109/IRDS.2002.1041519. URL http://dx.doi.org/10.1109/IRDS.2002.1041519.

[7] C. Bauckhage. *A structural framework for assembly modeling and recognition*. PhD thesis, Bielefeld University, Bielefeld, Germany, February 2002.

[8] The robot hall of fame. URL https://web.archive.org/web/20151117080257/http://www.robothalloffame.org/inductees/03inductees/unimate.html. Accessed: 2015-11-17.

[9] M. Muro and S. Andes. Robots seem to be improving productivity, not costing jobs. Harvard Business Review, June 2015. URL https://web.archive.org/web/20160213155252/https://hbr.org/2015/

06/robots-seem-to-be-improving-productivity-not-costing-jobs. Accessed: 2016-02-09.

[10] R. Cubek, W. Ertel, and G. Palm. High-level learning from demonstration with conceptual spaces and subspace clustering. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2592 – 2597. IEEE, May 2015. doi: 10.1109/ICRA.2015.7139548. URL http://dx.doi.org/10.1109/ICRA.2015.7139548.

[11] T. M. Anandan. Robotic assembly: Shrinking footprint, expanding market. Robotic Industries Association, August 2014. URL https://web.archive.org/web/20160213155814/http://www.robotics.org/content-detail.cfm/Industrial-Robotics-Industry-Insights/Robotic-Assembly-Shrinking-Footprint-Expanding-Market/content_id/4981. Accessed: 2016-02-09.

[12] J. Kulick, M. Toussaint, T. Lang, and M. Lopes. Active learning for teaching a robot grounded relational symbols. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1451 – 1457. AAAI Press, 2013. ISBN 978-1-57735-633-2. URL http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6706.

[13] O. Kroemer and J. Peters. Predicting object interactions from contact distributions. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, pages 3361 – 3367. IEEE, 2014. doi: 10.1109/IROS.2014.6943030. URL http://www.ias.tu-darmstadt.de/uploads/Publications/KroemerIROS2014.pdf.

[14] T. Jebara and R. Kondor. Bhattacharyya and expected likelihood kernels. In *Proceedings of the 16th Annual Conference on Learning Theory*, pages 57 – 71. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-40720-1. doi: 10.1007/978-3-540-45167-9_6. URL http://dx.doi.org/10.1007/978-3-540-45167-9_6.

[15] B. Rosman and S. Ramamoorthy. Learning spatial relationships between objects. *International Journal of Robotics Research (IJRR)*, 30(11):1328 – 1342, 2011. doi: 10.1177/0278364911408155. URL http://dblp.uni-trier.de/db/journals/ijrr/ijrr30.html#RosmanR11.

[16] K. Sjöö and P. Jensfelt. Learning spatial relations from functional simulation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1513 – 1519. IEEE, September 2011. doi: 10.1109/IROS.2011.6094780. URL http://dx.doi.org/10.1109/IROS.2011.6094780.

[17] H. Mosemann and F. Wahl. Automatic decomposition of planned assembly sequences into skill primitives. *IEEE Transactions on Robotics and Automation*, 17(5):709 – 718, 2001. doi: 10.1109/70.964670. URL http://dx.doi.org/10.1109/70.964670.

[18] A. Bourjault. Methodology of assembly automation: A new approach. In *Robotics and Factories of the Future '87*, editor R. Radharamanan, pages 37 – 45. Springer Berlin Heidelberg, 1988. ISBN 978-3-642-73892-0. doi: 10.1007/978-3-642-73890-6_6. URL http://dx.doi.org/10.1007/978-3-642-73890-6_6.

[19] T. De Fazio and D. E. Whitney. Simplified generation of all mechanical assembly sequences. *IEEE Journal of Robotics and Automation*, 3(6):640 – 658, December 1987. ISSN 0882-4967. doi: 10.1109/JRA.1987.1087132.

[20] L. E. Kavraki and M. N. Kolountzakis. Partitioning a planar assembly into two connected parts is NP-complete. *Information Processing Letters*, 55(3):159 – 165, 1995. ISSN 0020-0190. doi: http://dx.doi.org/10.1016/0020-0190(95)00083-O. URL http://www.sciencedirect.com/science/article/pii/0020019095000830.

[21] L. H. de Mello and A. C. Sanderson. Two criteria for the selection of assembly plans: Maximizing the flexibility of sequencing the assembly tasks and minimizing the assembly time through parallel execution of assembly tasks. *IEEE Transactions on Robotics and Automation*, 7(5):626 – 633, October 1991. ISSN 1042-296X. doi: 10.1109/70.97874. URL http://dx.doi.org/10.1109/70.97874.

[22] L. H. de Mello and A. C. Sanderson. A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Transactions on Robotics and Automation*, 7(2):228 – 240, 1991. doi: 10.1109/70.75905. URL http://dblp.uni-trier.de/db/journals/trob/trob7.html#MelloS91a.

[23] P. Gu and X. Yan. CAD-directed automatic assembly sequence planning. *International Journal of Production Research*, 33(11):3069 – 3100, 1995. doi: 10.1080/00207549508904862. URL http://dx.doi.org/10.1080/00207549508904862.

[24] D. Baldwin, T. Abell, M.-C. Lui, T. De Fazio, and D. E. Whitney. An integrated computer aid for generating and evaluating assembly sequences for mechanical products. *IEEE Transactions on Robotics and Automation*, 7(1): 78 – 94, February 1991. ISSN 1042-296X. doi: 10.1109/70.68072. URL http://dx.doi.org/10.1109/70.68072.

[25] A. J. Lambert. Determining optimum disassembly sequences in electronic equipment. *Computers & Industrial Engineering*, 43(3):553 – 575, 2002. ISSN 0360-8352. doi: http://dx.doi.org/10.1016/S0360-8352(02)00125-0. URL http://www.sciencedirect.com/science/article/pii/S0360835202001250.

[26] A. Koc, I. Sabuncuoglu, and E. Erel. Two exact formulations for disassembly line balancing problem and task precedence diagram construction using AND/OR graph. *IIE Transactions*, 41(10):866 – 881, August 2009. doi: 10.1080/07408170802510390. URL http://dx.doi.org/10.1080/07408170802510390.

[27] U. Thomas and F. Wahl. A system for automatic planning, evaluation and

execution of assembly sequences for industrial robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 1458 – 1464. IEEE, 2001. ISBN 0-7803-6612-3. doi: 10.1109/IROS.2001.977186. URL http://dx.doi.org/10.1109/IROS.2001.977186.

[28] S. H. Lee, H. K. Kim, and I. H. Suh. Incremental learning of primitive skills from demonstration of a task. In *Proceedings of the 6th International Conference on Human-Robot Interaction (HRI)*, pages 185 – 186. ACM, 2011. ISBN 978-1-4503-0561-7. doi: 10.1145/1957656.1957723. URL http://doi.acm.org/10.1145/1957656.1957723.

[29] S. Manschitz, J. Kober, M. Gienger, and J. Peters. Learning movement primitive attractor goals and sequential skills from kinesthetic demonstrations. *Robotics and Autonomous Systems*, 74, Part A:97 – 107, 2015. doi: 10.1016/j.robot.2015.07.005. URL http://dx.doi.org/10.1016/j.robot.2015.07.005.

[30] A. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. In *Advances in Neural Information Processing Systems 16 (NIPS)*, editors S. Thrun, L. K. Saul, and B. Schölkopf. MIT Press, 2003. ISBN 9780262201520. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84899019754&partnerID=40&md5=ab2909cc55a329782ed2f8dfa4864053.

[31] J. Morrow and P. K. Khosla. Manipulation task primitives for composing robot skills. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3354 – 3359. IEEE, 1997. ISBN 0-7803-3612-7. doi: 10.1109/ROBOT.1997.606800. URL http://dx.doi.org/10.1109/ROBOT.1997.606800.

[32] L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4(1):237 – 285, May 1996. ISSN 1076-9757. URL http://people.csail.mit.edu/lpk/papers/rl-survey.ps.

[33] S. Calinon. *Robot Programming by Demonstration: A Probabilistic Approach*. EPFL/CRC Press, 1 edition, 2009. ISBN 978-2-940222-31-5. URL http://calinon.ch/showPubli.php?publi=6001.

[34] A. Montebelli, F. Steinmetz, and V. Kyrki. On handing down our tools to robots: Single-phase kinesthetic teaching for dynamic in-contact tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 5628 – 5634. IEEE, May 2015. doi: 10.1109/ICRA.2015.7139987. URL http://dx.doi.org/10.1109/ICRA.2015.7139987.

[35] P. Kormushev, S. Calinon, and D. G. Caldwell. Imitation learning of positional and force skills demonstrated via kinesthetic teaching and haptic input. *Advanced Robotics*, 25(5):581 – 603, 2011. doi: 10.1163/016918611X558261. URL http://kormushev.com/papers/Kormushev_AdvancedRobotics_2011.pdf.

[36] S. Niekum, S. Osentoski, G. Konidaris, and A. Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, October 2012. doi: 10.1109/IROS.2012.6386006. URL http://lis.csail.mit.edu/pubs/niekum-iros12.pdf.

[37] M. Tykal. Optimizing programming by demonstration for in-contact task models by incremental learning. Master's thesis, Aalto University, Espoo, Finland, September 2015.

[38] S. Calinon and A. Billard. Incremental learning of gestures by imitation in a humanoid robot. In *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 255 – 262. ACM, March 2007. doi: 10.1145/1228716.1228751. URL http://lasa.epfl.ch/publications/uploadedFiles/Calinon-HRI2007.pdf.

[39] M. Muehlig, M. Gienger, S. Hellbach, J. J. Steil, and C. Goerick. Task-level imitation learning using variance-based movement optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1177 – 1184. IEEE, May 2009. ISBN 978-1-4244-2788-8. doi: 10.1109/ROBOT.2009.5152439. URL http://dx.doi.org/10.1109/ROBOT.2009.5152439.

[40] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1-5):11 – 73, February 1997. ISSN 0269-2821. doi: 10.1023/A:1006559212014. URL http://dx.doi.org/10.1023/A:1006559212014.

[41] J. Peters, S. Vijayakumar, and S. Schaal. Reinforcement learning for humanoid robotics. In *Proceedings of the 3rd IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 1 – 20. IEEE, 2003. URL http://www-clmc.usc.edu/publications/p/peters-ICHR2003.pdf.

[42] W. Khreich, E. Granger, A. Miri, and R. Sabourin. A survey of techniques for incremental learning of HMM parameters. *Information Sciences*, 197:105 – 130, August 2012. ISSN 0020-0255. doi: 10.1016/j.ins.2012.02.017. URL http://dx.doi.org/10.1016/j.ins.2012.02.017.

[43] S. Calinon, F. D'halluin, E. Sauser, D. Caldwell, and A. Billard. Learning and reproduction of gestures by imitation: An approach based on hidden Markov model and Gaussian mixture regression. *IEEE Robotics and Automation Magazine*, 17(2):44 – 54, 2010. ISSN 1070-9932. doi: 10.1109/MRA.2010.936947. URL http://infoscience.epfl.ch/record/147286/files/CalinonEtAl-RAM2010.pdf.

[44] S. Schaal, J. Peters, J. Nakanishi, and A. J. Ijspeert. Learning movement primitives. In *The International Symposium on Robotics Research (ISRR)*, pages 561 – 572. Springer Berlin Heidelberg, 2003. doi: 10.1007/11008941_60. URL http://dx.doi.org/10.1007/11008941_60.

[45] P. Pastor, M. Kalakrishnan, F. Meier, F. Stulp, J. Buchli, E. Theodorou, and S. Schaal. From dynamic movement primitives to associative skill memories. *Robotics and Autonomous Systems*, 61(4):351 – 361, 2013. doi: 10.1016/j.robot.2012.09.017. URL http://dx.doi.org/10.1016/j.robot.2012.09.017.

[46] S. Schaal. Dynamic movement primitives - a framework for motor control in humans and humanoid robotics. In *Adaptive Motion of Animals and Machines*, editors H. Kimura, K. Tsuchiya, A. Ishiguro, and H. Witte, pages 261 – 280. Springer Tokyo, 2006. ISBN 978-4-431-24164-5. doi: 10.1007/4-431-31381-8_23. URL http://dx.doi.org/10.1007/4-431-31381-8_23.

[47] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, 25(2):328 – 373, February 2013. ISSN 0899-7667. doi: 10.1162/NECO_a_00393. URL http://dx.doi.org/10.1162/NECO_a_00393.

[48] T. Asfour, P. Azad, N. Vahrenkamp, K. Regenstein, A. Bierbaum, K. Welke, J. Schröder, and R. Dillmann. Toward humanoid manipulation in human-centred environments. *Robotics and Autonomous Systems*, 56(1):54 – 65, 2008. ISSN 0921-8890. doi: http://dx.doi.org/10.1016/j.robot.2007.09.013. URL http://www.sciencedirect.com/science/article/pii/S0921889007001339.

[49] S. Ekvall and D. Kragic. Learning task models from multiple human demonstrations. In *The 15th IEEE International Symposium on Robot and Human Interactive Communication (ROMAN)*, pages 358 – 363. IEEE, September 2006. doi: 10.1109/ROMAN.2006.314460. URL http://dx.doi.org/10.1109/ROMAN.2006.314460.

[50] S. R. Ahmadzadeh, A. Paikan, F. Mastrogiovanni, L. Natale, P. Kormushev, and D. G. Caldwell. Learning symbolic representations of actions from human demonstrations. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2015. ISBN 978-1-4799-6923-4. doi: 10.1109/ICRA.2015.7139728. URL http://kormushev.com/papers/Ahmadzadeh_ICRA-2015.pdf.

[51] N. Abdo, H. Kretzschmar, and C. Stachniss. From low-level trajectory demonstrations to symbolic actions for planning. In *Proceedings of the ICAPS Workshop on Combining Task and Motion Planning for Real-World Applications (TAMPRA)*. AAAI Press, 2012. URL http://www.informatik.uni-freiburg.de/~kretzsch/pdf/abdo12tampra.pdf.

[52] K. Ikeuchi and T. Suchiro. Towards an assembly plan from observation. I. assembly task recognition using face-contact relations (polyhedral objects). In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2171 – 2177. IEEE, May 1992. doi: 10.1109/ROBOT.1992.219935. URL http://dx.doi.org/10.1109/ROBOT.1992.219935.

[53] H. Cheng and H. Chen. Learning from demonstration enabled robotic small part assembly. In *Proceedings of the 9th IEEE Conference on Industrial Electronics*

and Applications (ICIEA), pages 395 – 400. IEEE, June 2014. doi: 10.1109/ICIEA.2014.6931195. URL http://dx.doi.org/10.1109/ICIEA.2014.6931195.

[54] E. Bengoetxea. *Inexact Graph Matching Using Estimation of Distribution Algorithms*. PhD thesis, Ecole Nationale Supérieure des Télécommunications, Paris, France, December 2002.

[55] D. Koutra, A. Parikh, A. Ramdas, and J. Xiang. Algorithms for graph similarity and subgraph matching. Technical report, Carnegie Mellon University, 2011. URL https://www.cs.cmu.edu/~jingx/docs/DBreport.pdf.

[56] P. J. Kelly. A congruence theorem for trees. *Pacific Journal of Mathematics*, 7(1):961 – 968, 1957. URL http://projecteuclid.org/euclid.pjm/1103043674.

[57] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, pages 172 – 184. ACM, 1974. doi: 10.1145/800119.803896. URL http://doi.acm.org/10.1145/800119.803896.

[58] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: On approximating graph edit distance. *Proceedings of the Very Large Database (VLDB) Endownment*, 2(1):25 – 36, August 2009. ISSN 2150-8097. doi: 10.14778/1687627.1687631. URL http://dx.doi.org/10.14778/1687627.1687631.

[59] R. C. Wilson and P. Zhu. A study of graph spectra for comparing graphs and trees. *Pattern Recognition*, 41(9):2833 – 2841, 2008. ISSN 0031-3203. doi: http://dx.doi.org/10.1016/j.patcog.2008.03.011. URL http://www.sciencedirect.com/science/article/pii/S0031320308000927.

[60] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265 – 298, 2004. doi: 10.1142/S0218001404003228. URL http://www.worldscientific.com/doi/abs/10.1142/S0218001404003228.

[61] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, pages 117 – 128. IEEE Computer Society, 2002. doi: 10.1109/ICDE.2002.994702. URL http://dx.doi.org/10.1109/ICDE.2002.994702.

[62] G. Jeh and J. Widom. SimRank: A measure of structural-context similarity. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 538 – 543. ACM, 2002. ISBN 1-58113-567-X. doi: 10.1145/775047.775126. URL http://doi.acm.org/10.1145/775047.775126.

[63] S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*,

10(5):695 – 703, 1988. doi: 10.1109/34.6778. URL http://dblp.uni-trier.de/db/journals/pami/pami10.html#Umeyama88.

[64] T. Caelli and S. Kosinov. An eigenspace projection clustering method for inexact graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(4):515 – 519, 2004. doi: 10.1109/TPAMI.2004.1265866. URL http://doi.ieeecomputersociety.org/10.1109/TPAMI.2004.1265866.

[65] W.-J. Lee and R. Duin. An inexact graph comparison approach in joint eigenspace. In *Structural, Syntactic, and Statistical Pattern Recognition*, editors N. da Vitoria Lobo, T. Kasparis, F. Roli, J. Kwok, M. Georgiopoulos, G. Anagnostopoulos, and M. Loog, volume 5342 of *Lecture Notes in Computer Science*, chapter 8, pages 35 – 44. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-89688-3. doi: 10.1007/978-3-540-89689-0_8. URL http://dx.doi.org/10.1007/978-3-540-89689-0_8.

[66] F. R. K. Chung. *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92)*. American Mathematical Society, December 1996. ISBN 0821803158. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0821803158.

[67] L. Lovász. Eigenvalues of graphs, 2007. URL https://web.archive.org/web/20151206183610/http://www.cs.elte.hu/~lovasz/eigenvals-x.pdf. Accessed: 2015-12-06.

[68] F. Kummert, G. A. Fink, G. Sagerer, and E. Braun. Hybrid object recognition in image sequences. In *Proceedings of the 14th International Conference on Pattern Recognition (ICPR)*, volume 2, pages 1165 – 1170. IEEE, August 1998. doi: 10.1109/ICPR.1998.711903. URL http://dx.doi.org/10.1109/ICPR.1998.711903.

[69] H. Sundar, D. Silver, N. Gagvani, and S. Dickinson. Skeleton based shape matching and retrieval. In *Proceedings of the Shape Modeling International (SMI)*, pages 130 – 140. IEEE Computer Society, 2003. ISBN 0-7695-1909-1. doi: 10.1109/SMI.2003.1199609. URL http://portal.acm.org/citation.cfm?id=830339&dl=.

[70] A. Aldoma, F. Tombari, R. B. Rusu, and M. Vincze. OUR-CVFH – oriented, unique and repeatable clustered viewpoint feature histogram for object recognition and 6DOF pose estimation. In *Pattern Recognition: Proceedings of the Joint 34th DAGM and 36th OAGM Symposium*, pages 113 – 122. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-32717-9. doi: 10.1007/978-3-642-32717-9_12. URL http://dx.doi.org/10.1007/978-3-642-32717-9_12.

[71] N. Gordon, D. Salmond, and A. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings F – Radar and Signal Processing*, 140(2):107 – 113, April 1993. ISSN 0956-375X. doi: 10.1049/ip-f-2.1993.0015. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=210672.

[72] D. Fox. KLD-sampling: Adaptive particle filters. In *Advances in Neural Information Processing Systems 14 (NIPS)*, editors T. G. Dietterich, S. Becker, and Z. Ghahramani. MIT Press, 2002. ISBN 9780262042086. URL http://papers.nips.cc/paper/1998-kld-sampling-adaptive-particle-filters.pdf.

[73] F. Tombari, S. Salti, and L. Di Stefano. Unique signatures of histograms for local surface description. In *Proceedings of the 11th European Conference on Computer Vision (ECCV): Part III*, pages 356 – 369. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-15557-4. doi: 10.1007/978-3-642-15558-1_26. URL http://dl.acm.org/citation.cfm?id=1927006.1927035.

[74] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz. Aligning point cloud views using persistent feature histograms. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2008. ISBN 978-1-4244-2057-5. doi: 10.1109/IROS.2008.4650967. URL http://dx.doi.org/10.1109/IROS.2008.4650967.

[75] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu. Fast 3D recognition and pose using the viewpoint feature histogram. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2155 – 2162. IEEE, October 2010. ISBN 978-1-4244-6674-0. doi: 10.1109/IROS.2010.5651280. URL http://www.willowgarage.com/sites/default/files/Rusu10IROS.pdf.

[76] A. Aldoma, M. Vincze, N. Blodow, D. Gossow, S. Gedikli, R. B. Rusu, and G. Bradski. CAD-model recognition and 6DOF pose estimation using 3D cues. In *Proceedings of the IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 585 – 592. IEEE, 2011. ISBN 978-1-4673-0062-9. doi: 10.1109/iccvw.2011.6130296. URL http://dx.doi.org/10.1109/iccvw.2011.6130296.

[77] R. B. Rusu and S. Cousins. 3D is here: Point cloud library (PCL). In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1 – 4. IEEE, 2011. doi: 10.1109/ICRA.2011.5980567. URL http://dx.doi.org/10.1109/ICRA.2011.5980567.

[78] P. J. Besl and N. D. McKay. A method for registration of 3D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239 – 256, February 1992. ISSN 0162-8828. doi: 10.1109/34.121791. URL http://dx.doi.org/10.1109/34.121791.

[79] G. Kitagawa. Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, 5(1):1 – 25, 1996. ISSN 10618600. doi: 10.2307/1390750. URL http://dx.doi.org/10.2307/1390750.

[80] P. Lancaster and K. Salkauskas. Surfaces generated by moving least squares methods. *Mathematics of Computation*, 37(155):141 – 158, 1981. ISSN 00255718. doi: 10.2307/2007507. URL http://dx.doi.org/10.2307/2007507.

[81] F. M. ARTag, a fiducial marker system using digital techniques. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 590 – 596. IEEE Computer Society, 2005. doi: 10.1109/CVPR.2005.74. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-24644512963&partnerID=40&md5=8939c1cd75fcaef7ab1d2c71c59f7957.

[82] S. Nakano, G. Ueno, and T. Higuchi. Merging particle filter for sequential data assimilation. *Nonlinear Processes in Geophysics*, 14(4):395 – 408, 2007. doi: 10.5194/npg-14-395-2007. URL http://www.nonlin-processes-geophys.net/14/395/2007/.

[83] K. Collins, A. Palmer, and K. Rathmill. The development of a European benchmark for the comparison of assembly robot programming systems. In *Robot Technology and Applications*, editors K. Rathmill, P. MacConaill, P. O'Leary, and J. Browne, pages 187 – 199. Springer Berlin Heidelberg, 1985. ISBN 978-3-662-02442-3. doi: 10.1007/978-3-662-02440-9_18. URL http://dx.doi.org/10.1007/978-3-662-02440-9_18.